SPRING BOOT

SECURITY

Version: 2.0

Date: 04.2021

www.ivoronline.com

CONTENT

ABOUT THE BOOK

This is third Book in the series

Spring Boot - Quick Start

Spring Boot - Accessories

Spring Boot - Security

Content

Intention of this Book is to get you quickly started with Spring Boot Security by covering topics like: Authentication, Authorization, Roles, Authorities, Credentials, Login Form, Username, Password, CSRF, CORS, Remember Me, 2FA, JWT.

How to use this Book

Purpose of this Book is to learn functionalities covered by the tutorials in three steps:

1. Go through tutorials to learn specific functionalities
2. Go through Summary chapter which will help you organize what you have learned
3. Start practicing by using Demo Application, repeating tutorials or doing your work while using Summary chapter as a quick reference.

Standalone Tutorials

The core of this Book are standalone tutorials that explain different functionalities of Spring Boot Security. Each tutorial contains minimum amount of code and text needed to explain specific functionality. Such simple examples allow full understanding of the functionality without any unnecessary distractions. This approach allows students to grasp presented concepts in a fast and efficient manner. For each tutorial full code can be downloaded from GitHub to prevent wasting time typing or debugging the code.

Theoretical Background

Where needed tutorials are preceded by chapters focusing on theoretical background. This way reader can fully understand functionalities explained in the subsequent chapters. But such chapters are in minority and of secondary importance because the main focus is on code examples.

Demo Application

Book contains demo Application that shows how to combine some of the security features covered in the previous tutorials by focusing on JWT and Database Authentication.

WHY TUTORIALS?

"Things are only as complicated as they are badly explained"

Proper documentation is essential to avoid struggle and frustration when working with simple things that only seem complicated by not being properly documented and explained.

WHAT KIND OF TUTORIALS?

"Working example is worth thousand words"

Just like the picture is worth thousand words the same goes for the working example. Documentation in the form of working examples is proved to be the fastest and the most effective way of transferring knowledge. Sometimes an example is all you need to get the things done. And if there are some accompanying comments that explain what is going on even better. This approach is used in this book. This results in fast learning and the ability to apply tutorials when you need them in the spirit of Just In Time Support.

I wish you rapid learning!

1 Theory

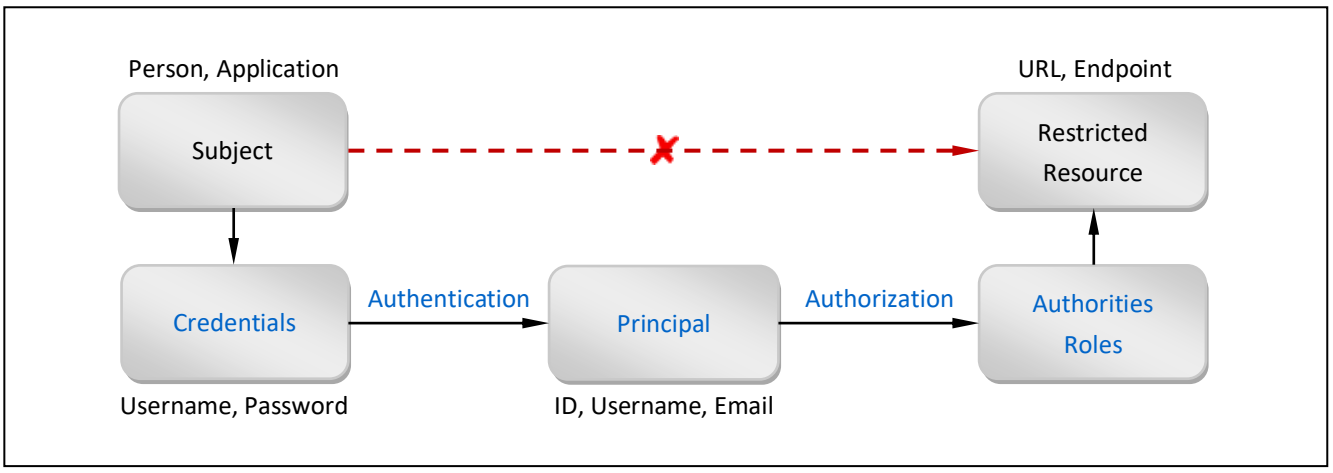
Info

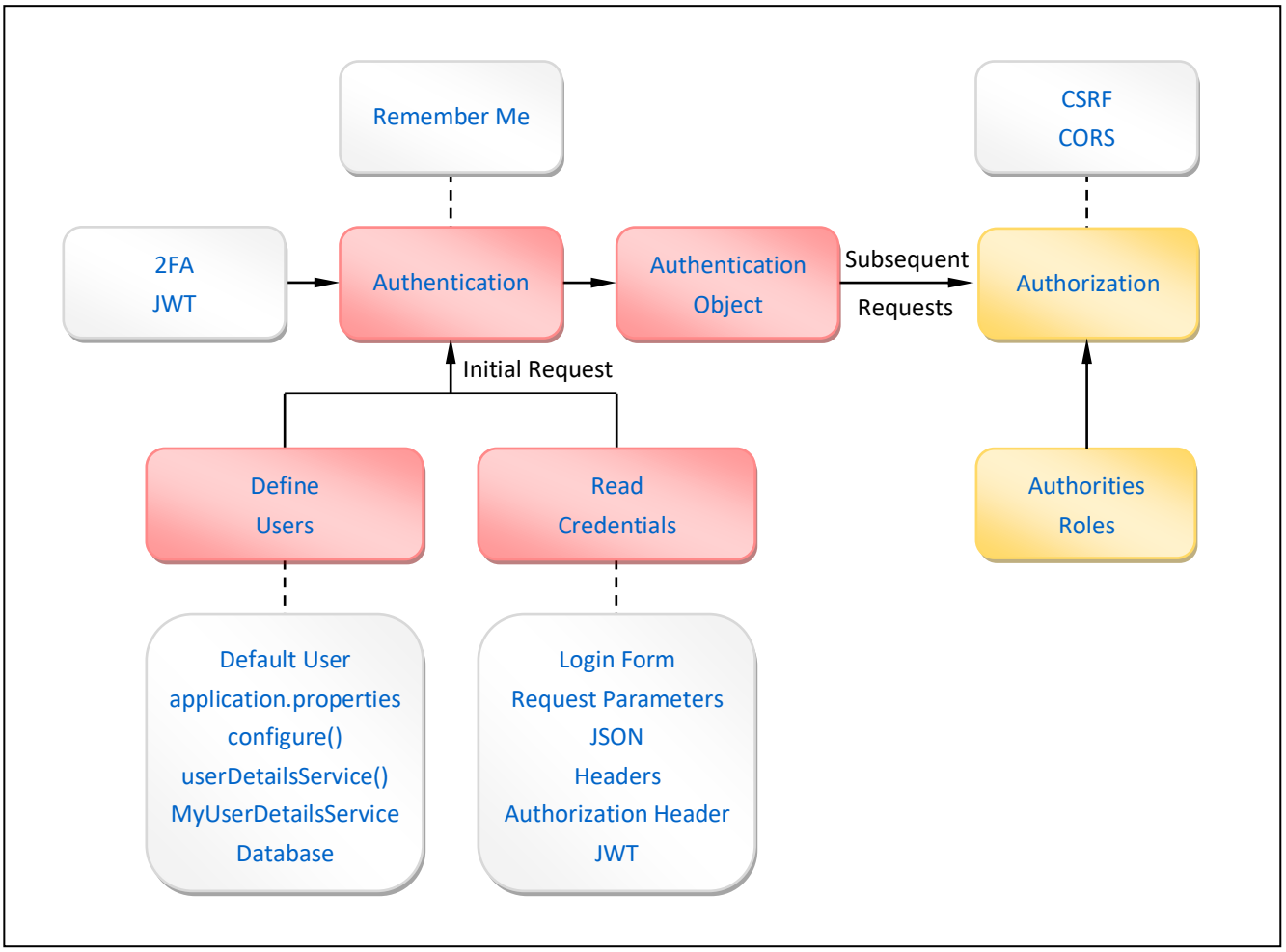
- Security is about preventing Subject to have direct access to Restricted Resource.
- Instead Subject has to
  - **Authenticate** himself (by providing Credentials: "Who are you?")
  - receive **Authorities** (which specify which Restricted Resources it can access: "What are you allowed to do?")

Main Terms

TERM	DESCRIPTION
Subject	Something that wants to access restricted Resource: Person, another Application
Credentials	Used to Authenticate Subject "Who are you?" (Username/Password, Token)
Authentication	Process of uniquely identifying Subject by using Credentials
Identity/Principal	Result of Authentication. Something that uniquely identifies Subject: ID, Username, Email, Phone
Authorization	Defines which restricted Resource are accessible to Principal: "What are you allowed to do?"
Authorities/Roles	Control access to Restricted Resources by assigning them to Principals and Restricted Resources.

Main Terms




Remember Me
CSRF
CORS
2FA
JWT
Authentication
Authentication
Object
Subsequent
Requests
Authorization
Initial Request
Define
Users
Read
Credentials
Authorities
Roles
Default User
application.properties
configure()
userDetailsService()
MyUserDetailsService
Database
Login Form
Request Parameters
JSON
Headers
Authorization Header
JWT

1.1 Authentication

Info

- **Authentication** answers question: "Who are you?"
  By using Credentials to identify Subject: Username, Password, Temporary Code.
- **Authentication** concerns itself with how to
  - store Users (application.properties, Class, Database)
  - allow Users to provide their Credentials (Login form, Authentication Header, JSON Body, Request Parameters)
  - compare Entered & Stored Credentials
- **Authentication** can be
  - **Database Authentication** if Users are store inside **DB**
  - **In-memory Authentication** if Users are store inside **Application**
    - **Default User** with autogenerated Password user/506e6f00-2b11-4036-96d6-74633e94da2d
    - **Single User** defined in `application.authorities` spring.security.user.name / password / roles
    - **Multiple Users** defined in `WebSecurityConfig` Class .username("myuser").password("mypass").roles("ADMIN")

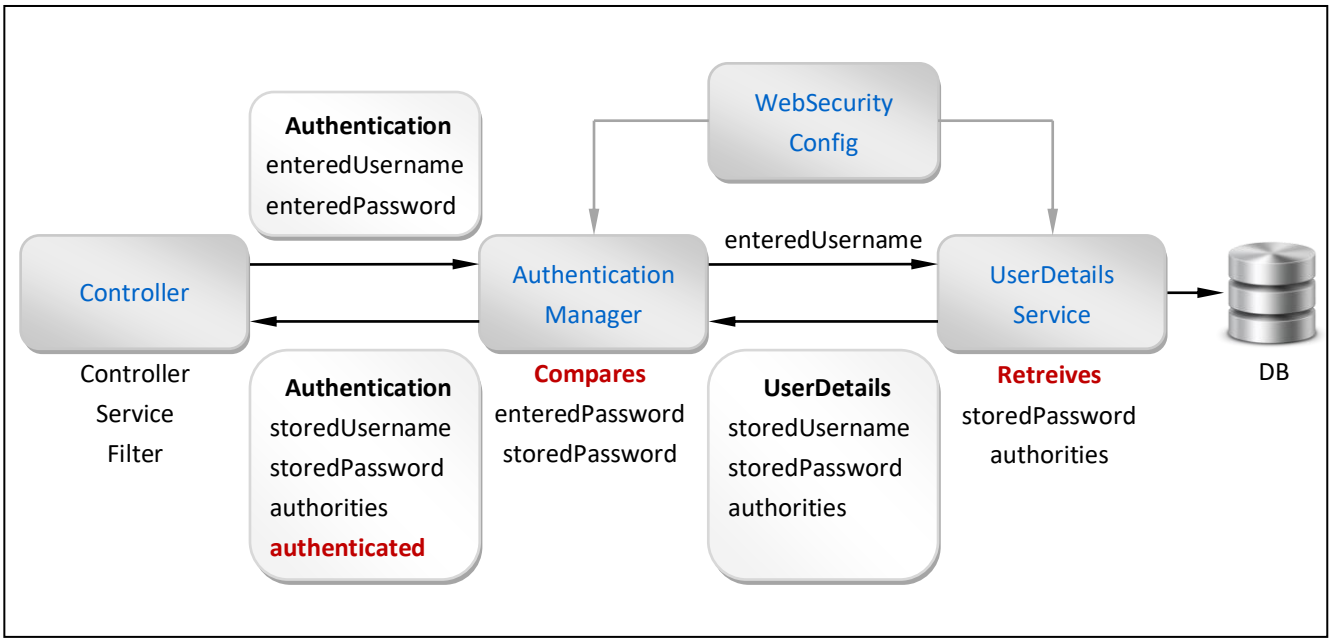
Authentication Classes & Objects

- Classes (that implement Methods needed for Authentication)
  - **UserDetailsService** uses **enteredUsername** to return **UserDetails** with **storedPassword** & **authorities**
  - **AuthenticationManager** uses **enteredUsername** & **storedPassword** from **UserDetails** to return **Authentication**
  - **WebSecurityConfig** is used to define Users, Password Encoder, add Authorities to Endpoints, ...
- Objects (that store data needed for Authentication)
  - **UserDetails** is **DTO** that transfers Stored Username, Password & Authorities from DB to **Authentication**
  - **Authentication** is used by Spring to control access to Endpoints using **Authorities** and **authenticated = true**

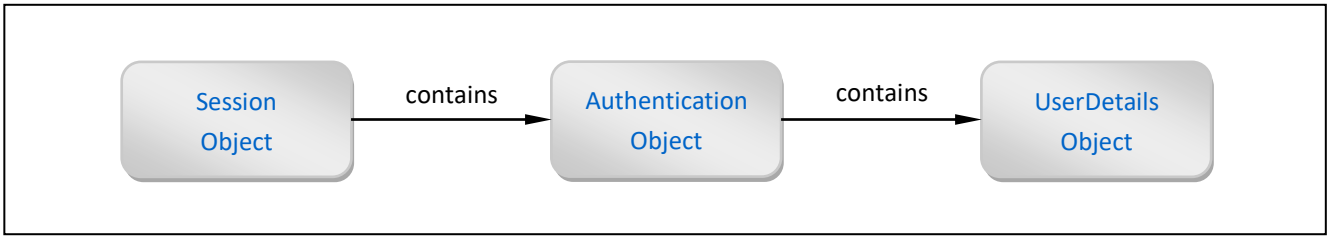
Authentication Process

- Code that initiates Authentication (Controller in below Schema)
  - reads **enteredUsername** and **enteredPassword** (from HTTP Request Parameters or Headers)
  - creates **Authentication** Object with **enteredUsername** and **enteredPassword**
  - calls **AuthenticationManager authenticate()** Method with **Authentication** Object as Input Parameter
  - stores returned **Authentication** Object in Context (so that Authorities can be used to access Restricted Resources)
- **AuthenticationManager**
  - receives **Authentication** Object as Input Parameter (with enteredUsername & enteredPassword)
  - calls **UserDetailsService** with **enteredUsername**
  - from **UserDetailsService** it gets **UserDetails** Object with **storedPassword** and **authorities**
  - compares **enteredPassword** with **storedPassword**
  - If Passwords match User is considered Authenticated and **AuthenticationManager** returns **Authentication** Object with
    - **storedUsername**
    - **storedPassword**
    - **authorities** (taken from UserDetails Object)
    - **authenticated = true**

Authentication Process



Main Authentication Objects



1.1.1 Identity/Principal

Info

- Once the Subject is Authenticated unique **Identity/Principal** is assigned/stored **to uniquely identify Subject**.
  For instance if Subject made a HTTP Request, Application can create Session in which it stores unique **Identity/Principal** so that the Application knows which Subject is sending subsequent HTTP Requests.
- For instance two Subjects/Users/Persons with the name Jack Carpenter might be logged in at the same time.
  Since they were logged in using different Credentials (combination of Username and Password) Application was able to Authenticate them as two different Subjects with two different IDs which serve as their Identities/Principals
  - First Jack Carpenter holds ID = 100 in Application's DB
  - Second Jack Carpenter holds ID = 200 in Application's DB

1.1.2 Credentials vs Principal

Info

- Confusion sometimes arises from the fact that the same thing can sometimes have multiple roles.
  - For instance **Username** can be part of **Credentials** to allow Authentication.
  - But if Usernames are unique across the Application then Username can also be used as **Identity/Principal** (something that uniquely identifies Subject inside the Application)
- Some Applications allow Users to have the same Usernames (like Facebook).
  So in such Applications Identity/Principal has to be something else like: ID, Email Address, Phone Number.

1.1.3 UserDetails Object

Info

- During Authentication, **UserDetails** DTO Object is used to **transfer** User data from Database related to Entered Username.
- **UserDetails** Object will contain: Username, Stored Password, Authorities.
- If there was no User found for Entered Username then **Exception** is thrown.

1.1.4 Authentication Object

Info

- **Authentication** Object contains data of currently signed in User: Username, Stored Password, Authorities.
- And an additional Property which says if User was properly Authenticated
  - if User was found in DB and Password was correct **authentication = true**
  - if User was found in DB but Password was wrong **authentication = false**
- **Authentication** Manager compares Entered Password with Stored Password from `UserDetails` Object.
  If they match **Authentication** Object is created with `UserDetails` Object as one of its Properties.

1.1.5 Session Object

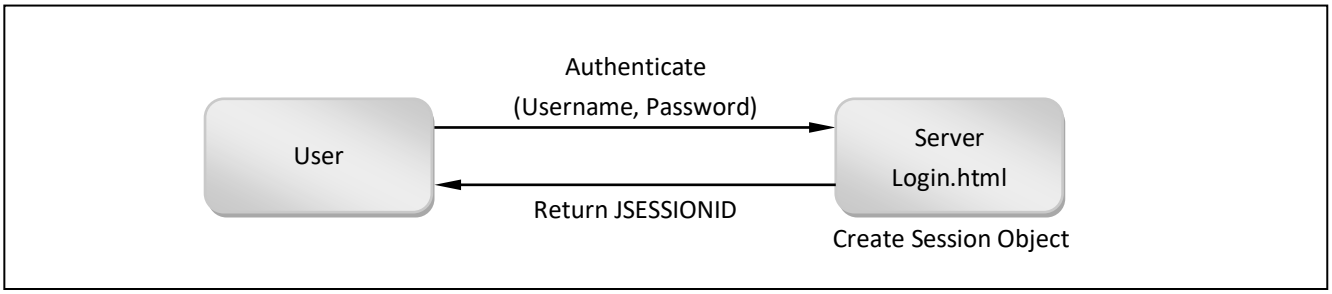
Info

- **Session Object**
  - is Java Object
  - is stored on the Server (either in the Memory or in the Database)
  - is used to track different Users (different Session Object is assigned to each User)
  - contains User related data (Authentication Object)
- **JSESSIONID**
  - is Cookie
  - is received from Server (after successful Authentication)
  - is stored in the Browser
  - is sent to Server with every HTTP Request
  - contains ID that uniquely identifies User's Session Object
- **JSESSIONID and Session Object are used to track different Users between subsequent HTTP Requests** (and their data).
  - This way Server can for instance know if User has already Authenticated (by providing valid Username and Password) to allow User to access restricted Resources in subsequent HTTP Requests.
  - For instance upon successful Authentication, Server can store `authenticated=true` and `userRole=`**ADMIN** in User's Session Object and return JSESSIONID Cookie to uniquely identify created Session Object. Then when that User sends next HTTP Request together with stored JSESSIONID Cookie, Server will use JSESSIONID to retrieve User's Session Object and will allow him to access restricted Resource based on User's Role `ADMIN`.
  - For another User Server would create different Session Object and JSESSIONID Cookie to identify it. For that User upon successful Authentication Server can store `authenticated=true` and `userRole=`**USER** in User's Session Object. Then when User sends next HTTP Request Server will allow him to access restricted Resource based on User's Role `USER`.

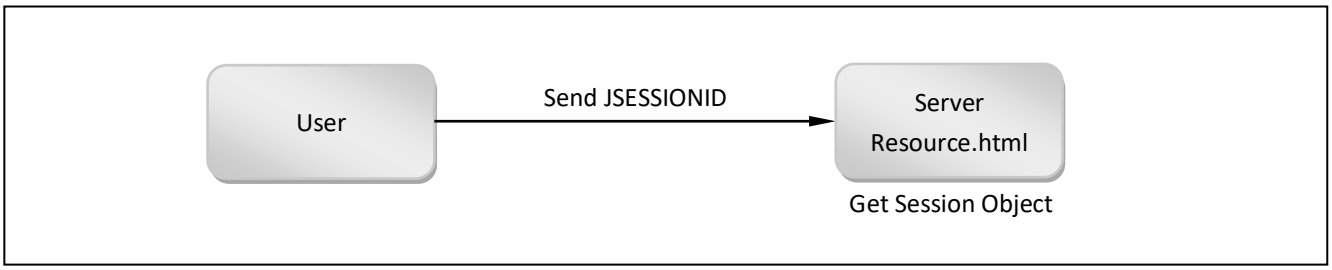
Session Object Example

```
id            = JSESSIONID
userID        = 5248
userName      = John
userRole      = USER
authenticated = true
```

***Initial** HTTP Request*



***Subsequent** HTTP Requests*



1.2 Authorization

Info

- **Authorization** answers question: "What are you allowed to do?"
  Authorization assigns Authorities & Roles to both Users and Endpoints.
  This is how access to Restricted Resources is controlled.
  User can only access Resources that have the same Authorities & Roles as him.
- **Authorization** can be
  - **Role** Based `@Secured("ADMIN")`
  - **Authorities** Based `@PreAuthorize("hasAuthority(book.create)")`
  - **Custom** Based `@PreAuthorize("@authenticationService.authenticate(authentication)")`

Custom Based Authorization

- Custom Authorization is based on calling **Custom Methods** which should return **Boolean**
  - If Method returns **true** User will be allowed to access endpoint
  - If Method returns **false** User will be forbidden to access endpoint
- You can combine **Authorities** and **Custom Methods** to control access to endpoints.
- **Custom Authorization** is usually combined with **Custom Users** since they have additional Custom Properties that can be used to filter access to endpoints.

1.2.1 Authorities vs Roles

Info

- Spring Security is based on **Authorities**.
- User is first Authenticated through combination of its Username and Password.
- Then Spring creates UserDetails Object that has list of Authorities that are assigned to that user (from DB for instance).
- These Authorities are then used inside the Controller to control access to the Endpoints.

Roles

- **Roles** are just Authorities whose name starts with prefix **ROLE_**.
  Inside the Controller you check for Roles and Authorities by using Annotations and Methods.
  Some of these Annotations and Methods are specialized to be used only to look for Roles
  - @Secured("ADMIN") is specialized Annotations that looks for Authority named ROLE_ADMIN
  - hasRole() in @PreAuthorize("hasRole('ADMIN')") is specialized Method that looks for Authority named ROLE_ADMIN
    This Method has the same effect as using @PreAuthorize("hasAuthority('ROLE_ADMIN')").

Profiles

- You can also customize Authorization by using **Profiles as containers for Authorities**.
  In DB you assign Profiles to Users and then you assign Authorities to these Profiles (using @ManyToMany Relationship)
  Then when creating UserDetails Object you load it with Authorities that are related to Profile of Authenticated User.
- You can also **add Profile to list of User Authorities**.
  This might be useful to detect from which Profile those Authorities came from.
  For instance both ADMIN and USER Profiles might have book.read Authority
  - But if book.read Authority came from ADMIN Profile then Admin can read any book (no other checks are necessary).
  - But if book.read Authority came from USER Profile then User can only read his own books.
    In that case an additional check is needed through **Custom Authorization** where we compare ID of current User with User ID assigned to the Book, and allow access to Endpoint only if these two IDs match (if User owns the Book).

2 Main Terms

Info

- Following tutorials explain main terms related to Spring Boot Security.

2.1 Define Users

Info

- Following tutorials show different ways of defining Users and their Roles and Authorities.

Default User

- With Default User your Application can have a **single** User.
- This means that only one User can log into your application to access its endpoints but
  - you can't define **Username** (it is always **user**)
  - you can't define **Password** (it is **auto-generated** every time you start Application)

Console

```
Using generated security password: 506e6f00-2b11-4036-96d6-74633e94da2d
```

User defined in application.properties

- With User defined in application.properties your Application can again only have a **single** User.
- This means that only one User can log into your application to access its endpoints but
  - this time you can define **Username** spring.security.user.name = myuser
  - this time you can define **Password** spring.security.user.password = mypassword
  - this time you can define **Role** spring.security.user.roles = USER
- Using this setup you are good to go with **Role Based Security**.
- But if you want you can expand application.properties with custom Properties to specify Authorities for each Role.
  This way you can implement **Authority Based Security**.
  But it will be very limited since your Application could still only have one User at the time.
  Although each time you start Application you could change its Role and therefore its related Authorities.

application.properties

```
spring.security.user.name     = myuser
spring.security.user.password = mypassword
spring.security.user.roles    = USER, ADMIN
```

Users defined in WebSecurityConfig.java

- Users can be defined programmatically with API in WebSecurityConfig.java Class.
- The way you can have **multiple** Users with different Username, Password, Roles and Authorities.
- Class should extend WebSecurityConfigurerAdapter and override either
  - userDetailsService()
  - configure()

WebSecurityConfig.java

```
@Configuration
@EnableWebSecurity
public class WebSecurityConfig extends WebSecurityConfigurerAdapter {

  @Bean
  @Override
  protected UserDetailsService userDetailsService() {

    //CREATE ADMIN
    UserDetails admin = User.withDefaultPasswordEncoder()
      .username("myadmin").password("myadminpassword").roles("ADMIN").build();

    //CREATE USER
    UserDetails user = User.withDefaultPasswordEncoder()
        .username("myuser").password("myuserpassword").roles("USER").build();

    //ADD USERS
    return new InMemoryUserDetailsManager(admin, user);

  }

}
```

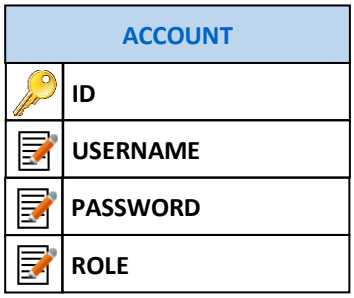
WebSecurityConfig.java

```
@Configuration
@EnableWebSecurity
public class WebSecurityConfig extends WebSecurityConfigurerAdapter {

  @Override
  protected void configure(AuthenticationManagerBuilder auth) throws Exception {
    auth.inMemoryAuthentication().withUser("myadmin").password("{noop}myadminpassword").roles("ADMIN");
    auth.inMemoryAuthentication().withUser("myuser" ).password("{noop}myuserpassword" ).roles("USER" );
  }

}
```

Users defined in DB

- Defining Users in Database is the most common way.
- This also allows you to have **multiple** Users with different Username, Password, Roles and Authorities.
- Users can be assigned a Profile which has associated Authorities that should be given to User.

DB Schema



MyUserDetailsService.java

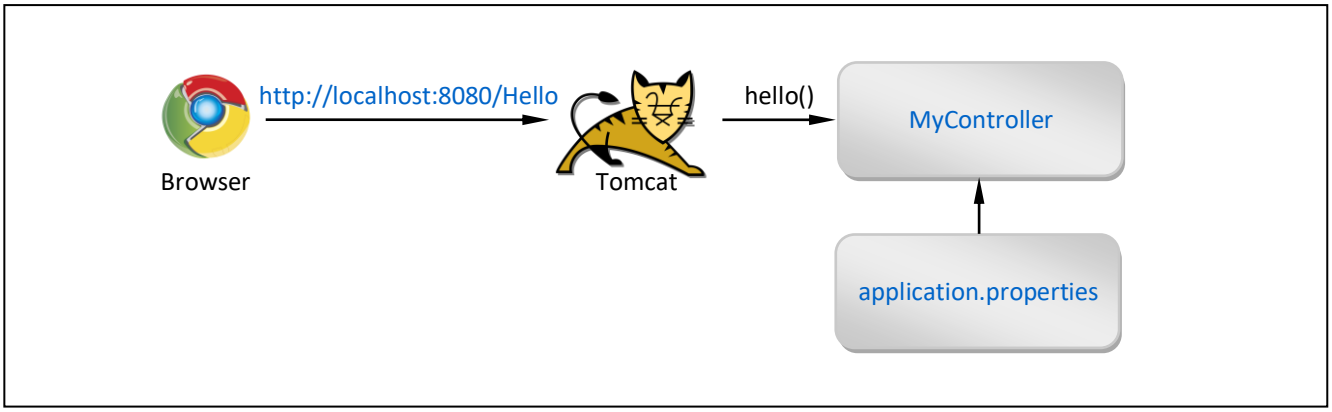
```
@Service
public class MyUserDetailsService implements UserDetailsService {

  @Autowired AccountRepository accountRepository;

  @Override
  public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {

    //GET ACCOUNT FROM DB
    Account account  = accountRepository.findByUsername(username);
    String  password = account.password;

    //CREATE AUTHORITIES
    List<GrantedAuthority> authorities = new ArrayList<GrantedAuthority>();
                           authorities.add(new SimpleGrantedAuthority(account.role));

    //CREATE USER
    User user = new User(username, password, authorities);

    //RETURN USER
    return user;

  }

}
```

2.1.1 application.properties

Info **[G]**

- This tutorial shows how to use **application.properties** to define single User with Roles.
- We will use
  - **application.properties** (to define single User with Roles)
  - @Secured("ROLE_ADMIN") (to add Authorities to Endpoint /Hello)
  - Default Login Form (to enter Credentials)
- You can't specify multiple Users and you can't specify Authorities (because prefix "ROLE_" is implied).
  Tutorial Authorities - application.properties shows custom approach for specifying Authorities through Profiles.
- We will use **@Secured("ROLE_ADMIN")** to assign Role to Endpoint to demonstrate that User's and Endpoint's Roles must match for User to have access to Endpoint. If Endpoint has no Roles then it is enough for User to just be logged in.
- To enable @Secured Annotation we need to add **@EnableGlobalMethodSecurity(securedEnabled = true)** to some Configuration Class like the main Application Class SpringbootSecurityUserApplicationpropertiesApplication.

Application Schema *[Results]*



Spring Boot Starters

GROUP	DEPENDENCY	DESCRIPTION
Web	Spring Web	Enables: @RestController, @RequestMapping, Tomcat Server
Security	Spring Security	Enables: Spring Security

Procedure

- Create Project: springboot_security_user_applicationproperties (add Spring Boot Starters from table)
- Edit File: application.properties (contains Username, Password, Roles)
- Edit Class: SpringbootSecurityUserApplicationpropertiesApplication.java (@EnableGlobalMethodSecurity)
- Create Package: controllers (inside main package)
  - Create Class: MyController.java (inside controllers package)

application.properties

```
# SECURITY
spring.security.user.name     = myuser
spring.security.user.password = myuserpassword
spring.security.user.roles    = USER, ADMIN
```

SpringbootSecurityUserApplicationpropertiesApplication.java

```
package com.ivoronline.springboot_security_user_applicationproperties;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.security.config.annotation.method.configuration.EnableGlobalMethodSecurity;

@SpringBootApplication
@EnableGlobalMethodSecurity(securedEnabled = true)
public class SpringbootSecurityUserApplicationpropertiesApplication {

  public static void main(String[] args) {
    SpringApplication.run(SpringbootSecurityUserApplicationpropertiesApplication.class, args);
  }

}
```

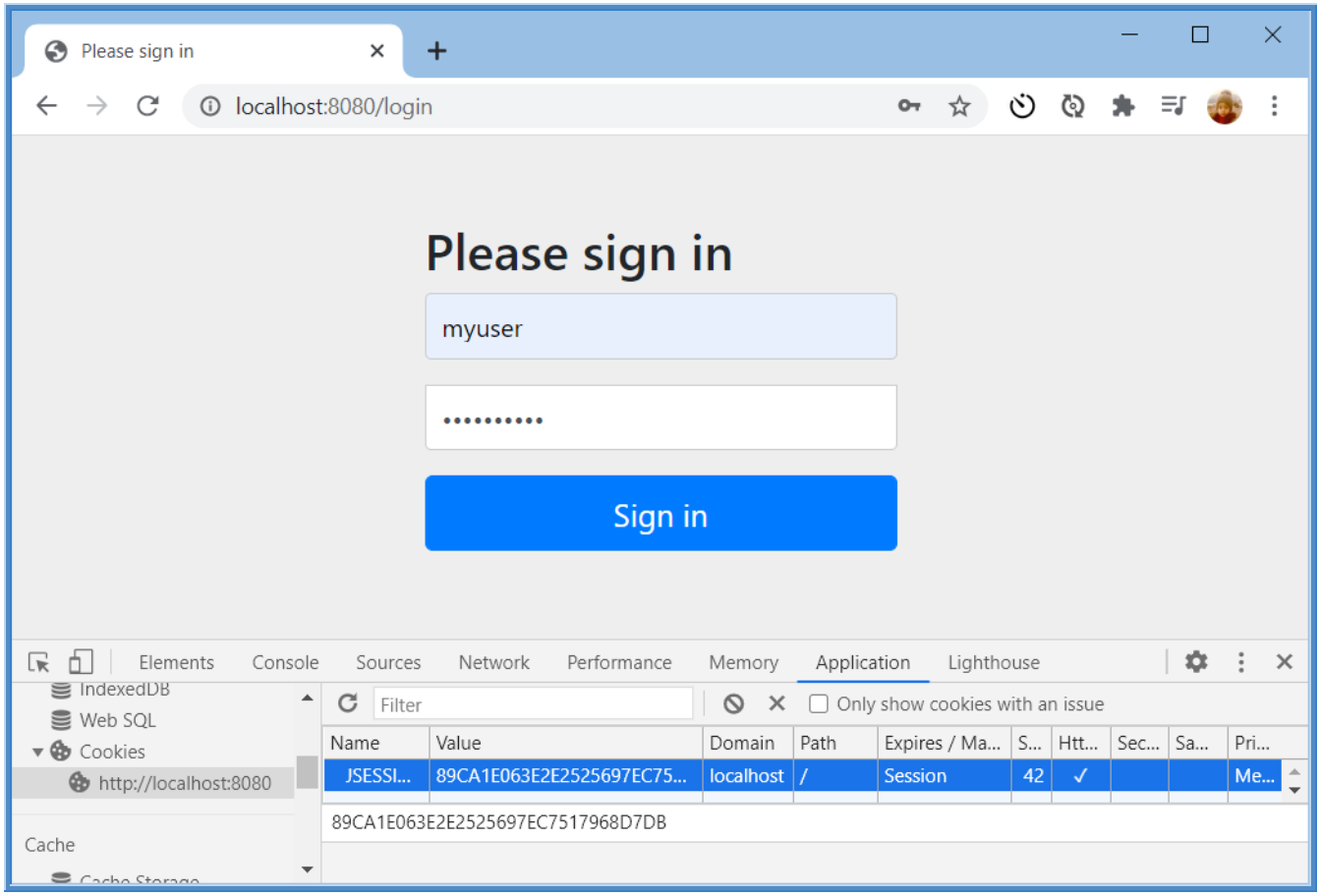
MyController.java

```
package com.ivoronline.springboot_security_user_applicationproperties.controllers;

import org.springframework.security.access.annotation.Secured;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class MyController {

  @Secured("ROLE_ADMIN")
  @RequestMapping("/Hello")
  public String hello() {
    return "Hello from Controller";
  }

}
```

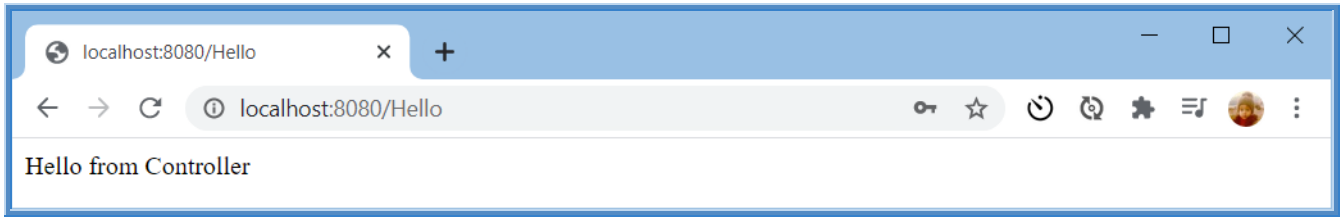
Results

- http://localhost:8080/Hello
- You get redirected to http://localhost:8080/login
  - Username: **myuser**
  - Password: **myuserpassword**
  - Sign in
- You get redirected back to http://localhost:8080/Hello
- http://localhost:8080/logout
  - Log Out

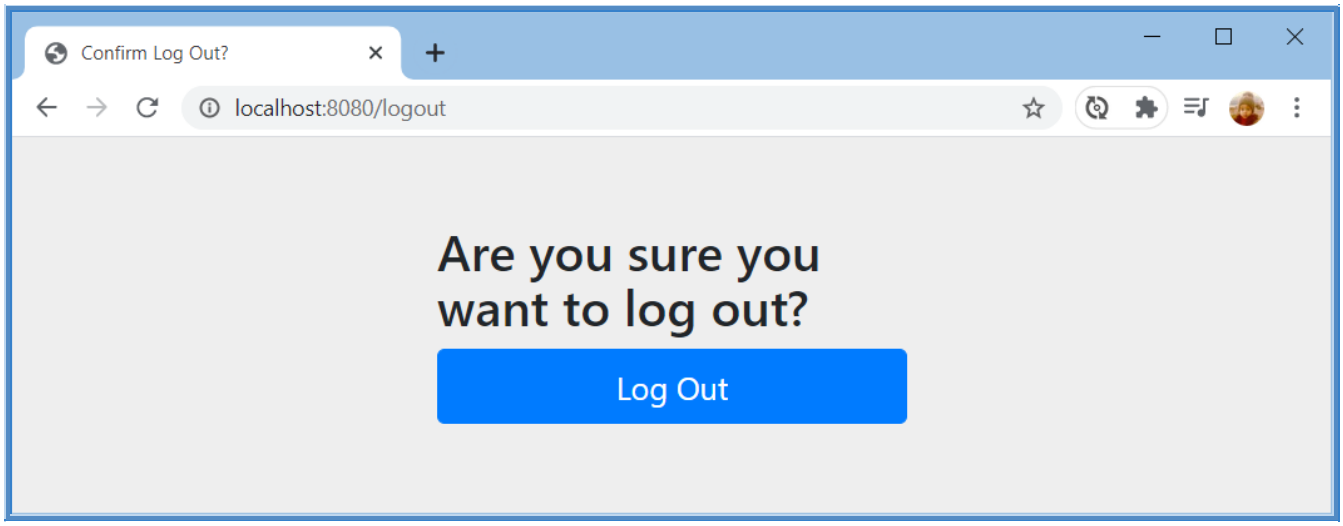
http://localhost:8080/login *(JSESSIONID Cookie is stored in the Browser)*



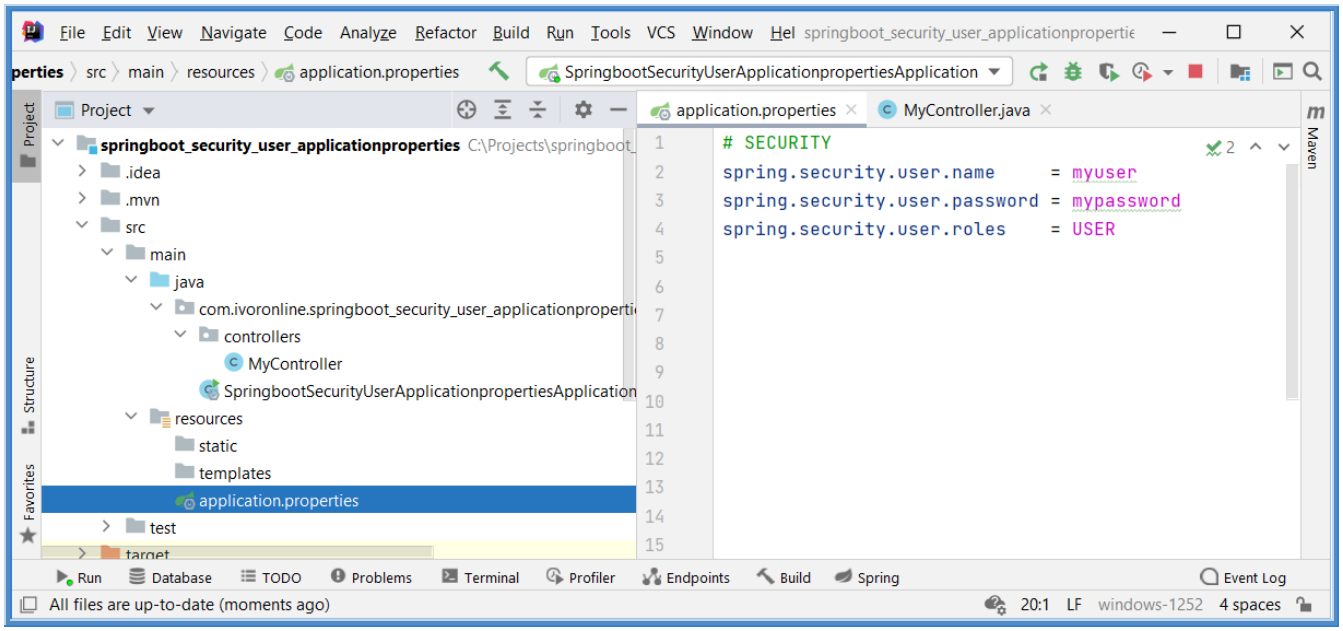
Redirects to http://localhost:8080/Hello



http://localhost:8080/logout *(Redirects to http://localhost:8080/login)*



Application Structure



pom.xml

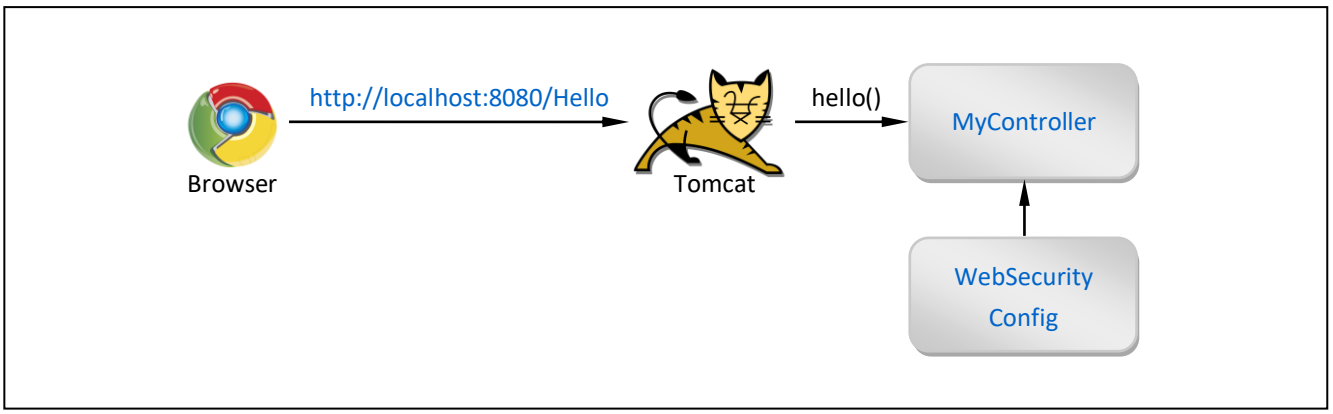
```
<dependencies>

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

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

</dependencies>
```

2.1.2 WebSecurityConfig - configure()

Info [G]

- This tutorial shows how to use **configure(Authentication...)** to define multiple Users inside WebSecurityConfig.
- We will use
  - **configure(Authentication...)** (to define multiple Users inside WebSecurityConfig)
  - @Secured("ROLE_ADMIN") (to add Authorities to Endpoint /Hello)
  - Default Login Form (to read Credentials)
- Here we are specifying multiple Users by (application.properties can only specify single User)
  - creating Class that extends WebSecurityConfigurerAdapter Class
  - overriding Method configure() (AuthenticationManagerBuilder auth)
- We will use **@Secured("ROLE_ADMIN")** to assign Role to Endpoint to demonstrate that User's and Endpoint's Roles must match for User to have access to Endpoint. If Endpoint has no Roles then it is enough for User to just be logged in. To enable @Secured Annotation we need to add **@EnableGlobalMethodSecurity(securedEnabled = true)** to some Configuration Class like WebSecurityConfig.java.

Application Schema *[Results]*



Spring Boot Starters

GROUP	DEPENDENCY	DESCRIPTION
Web	Spring Web	Enables @RequestMapping and Tomcat
Security	Spring Security	Enables Spring Security

Users

USERNAME	PASSWORD	ROLES
myadmin	myadminpassword	ADMIN
myuser	myuserpassword	USER

Procedure

- Create Project: springboot_security_class2 (add Spring Boot Starters from the table)
- Create Package: config (inside main package)
  - Create Class: WebSecurityConfig.java (inside config package)
- Create Package: controllers (inside main package)
  - Create Class: MyController.java (inside controllers package)

WebSecurityConfig.java

```
package com.ivoronline.springboot_security_class2.config;

import org.springframework.context.annotation.Configuration;
import org.springframework.security.config.annotation.authentication.builders.AuthenticationManagerBuilder;
import org.springframework.security.config.annotation.method.configuration.EnableGlobalMethodSecurity;
import org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;

@Configuration
@EnableGlobalMethodSecurity(securedEnabled = true)
public class WebSecurityConfig extends WebSecurityConfigurerAdapter {

  //=========================================================================
  // CONFIGURE (Auth...)
  //=========================================================================
  @Override
  protected void configure(AuthenticationManagerBuilder auth) throws Exception {
    auth.inMemoryAuthentication().withUser("myadmin").password("{noop}myadminpassword").roles("ADMIN");
    auth.inMemoryAuthentication().withUser("myuser" ).password("{noop}myuserpassword" ).roles("USER" );
  }

}
```

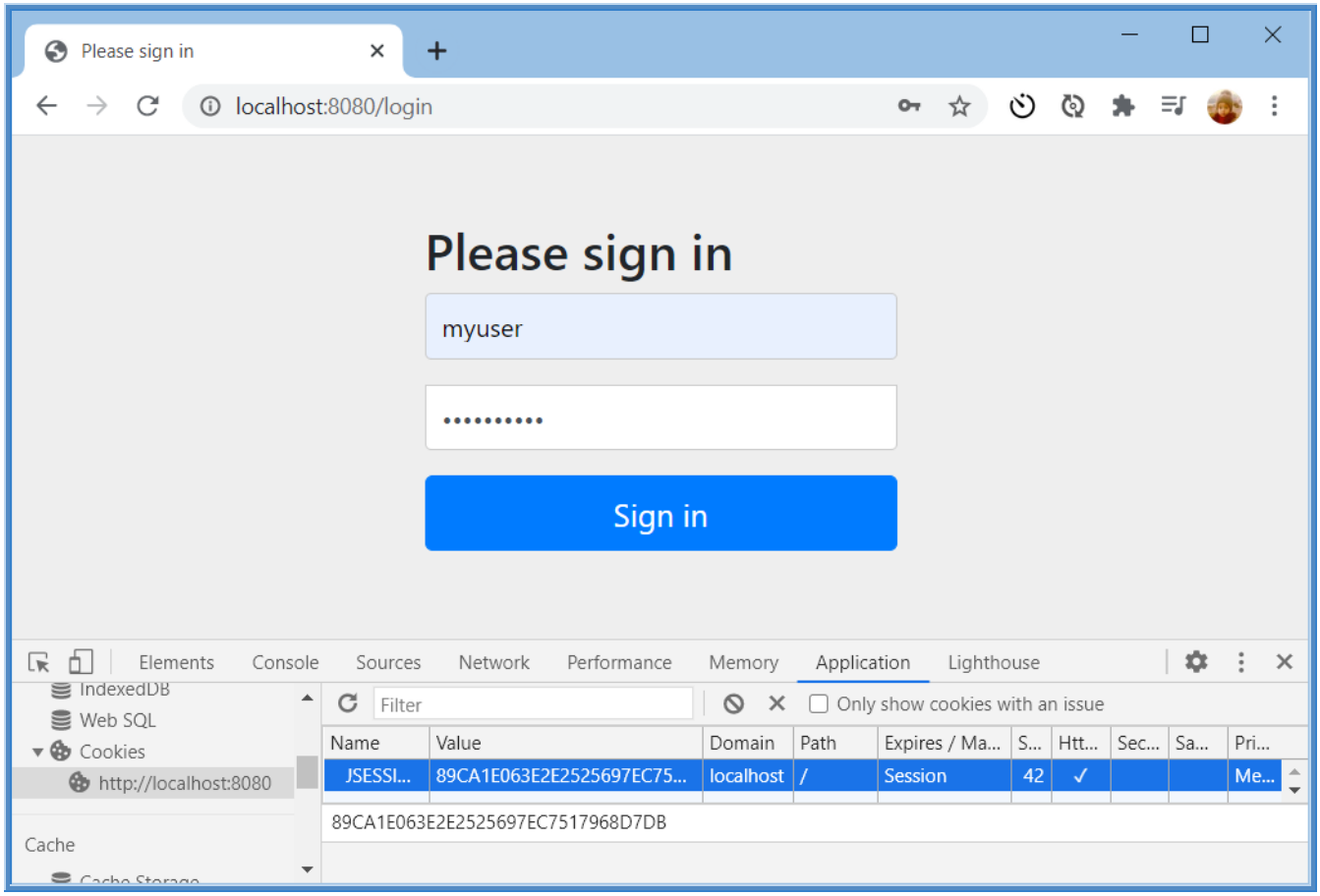
MyController.java

```
package com.ivoronline.springboot_security_class2.controllers;

import org.springframework.security.access.annotation.Secured;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class MyController {

  //=========================================================================
  // HELLO
  //=========================================================================
  @Secured("ROLE_ADMIN")
  @RequestMapping("/Hello")
  public String hello() {
    return "Hello from Controller";
  }

}
```

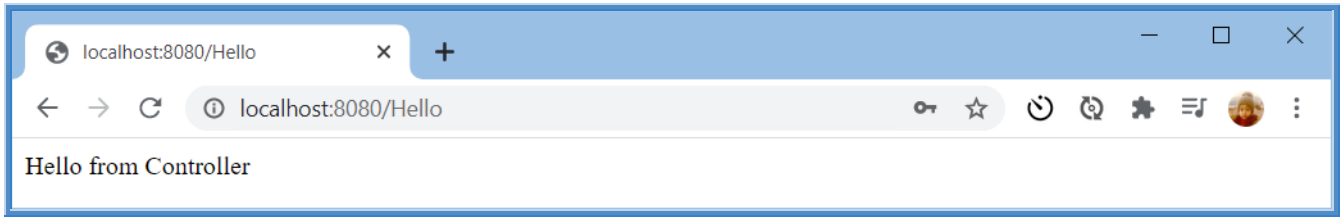
Results

- http://localhost:8080/Hello
- You get redirected to http://localhost:8080/login
  - Username: **myuser**
  - Password: **myuserpassword**
  - Sign in
- You get redirected back to http://localhost:8080/Hello
- http://localhost:8080/logout
  - Log Out

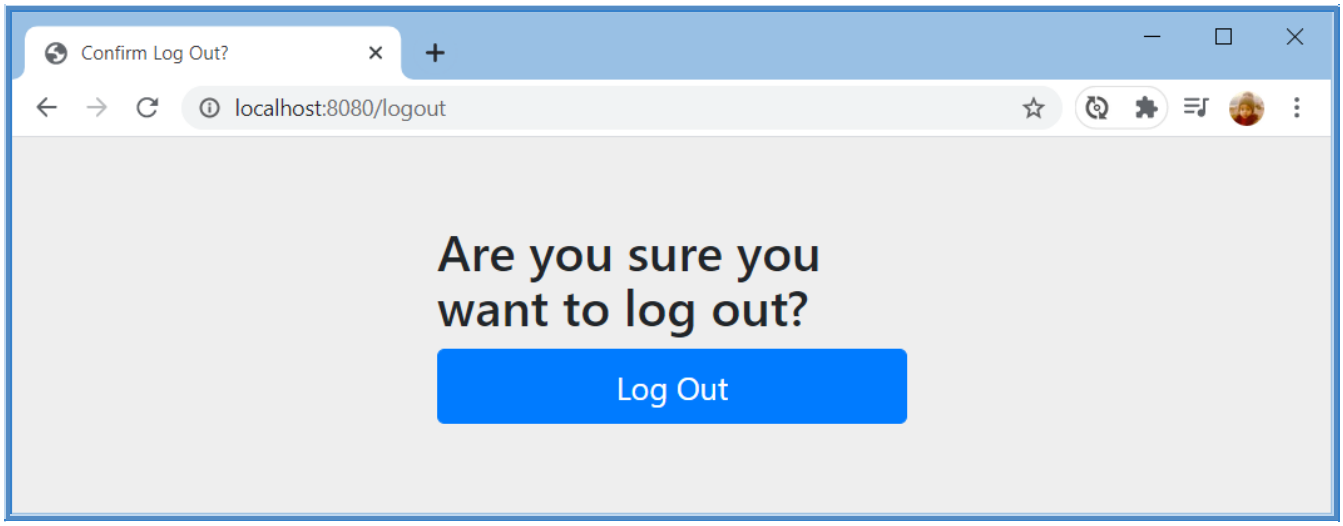
http://localhost:8080/login *(JSESSIONID Cookie is stored in the Browser)*



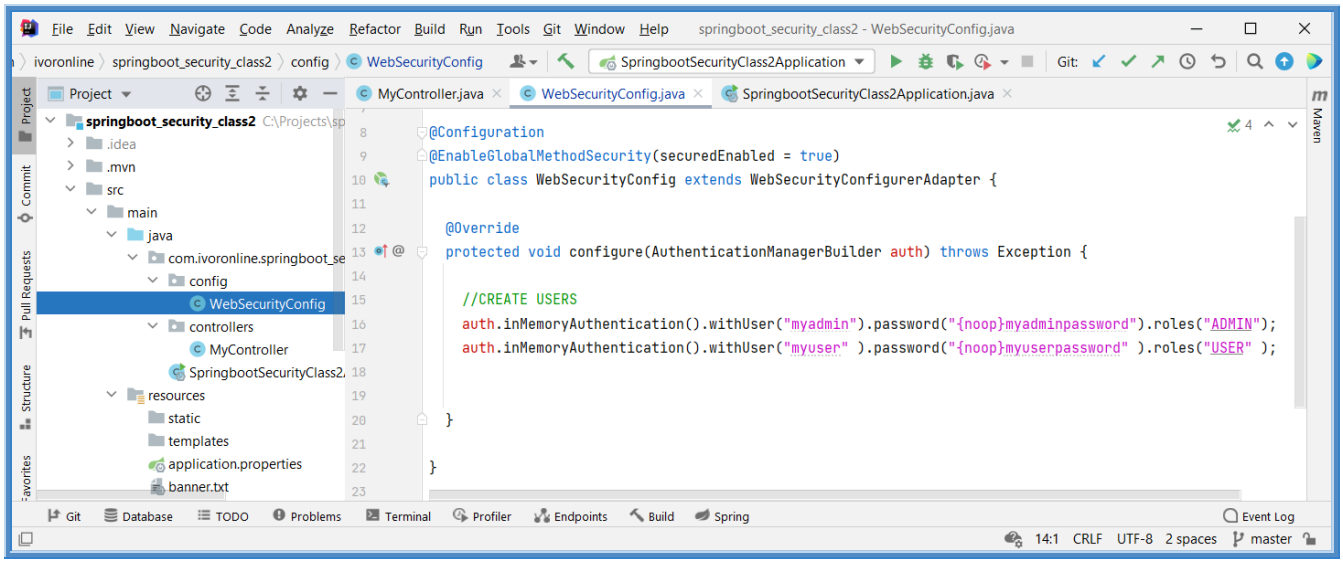
Redirects to http://localhost:8080/Hello



http://localhost:8080/logout *(Redirects to http://localhost:8080/login)*



Application Structure



pom.xml

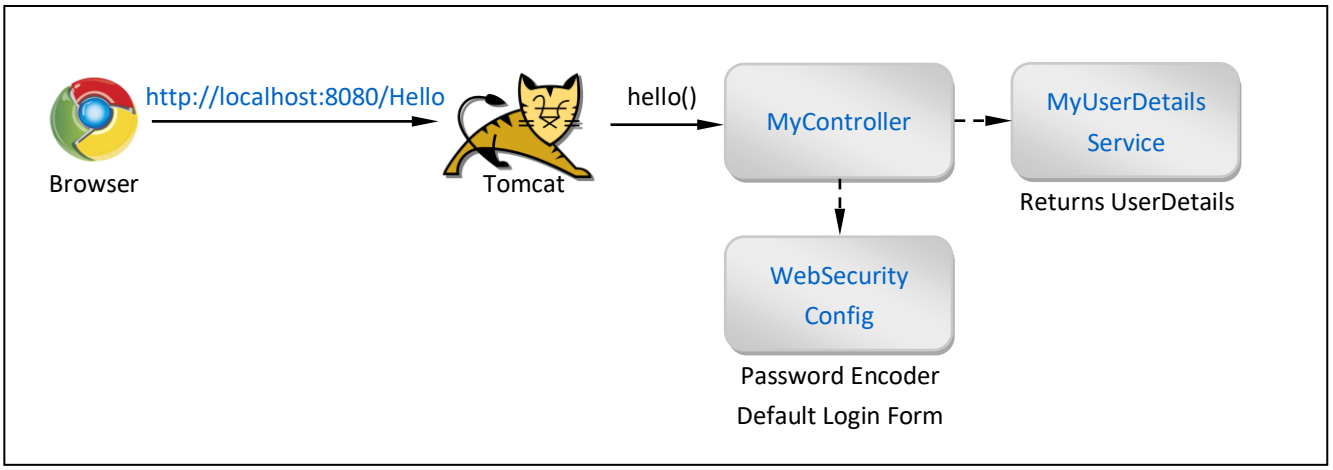
```
<dependencies>

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

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

</dependencies>
```

2.1.3 MyUserDetailsService - Hard Coded Users

Info [G]

- This tutorial shows how to use **MyUserDetailsService** to return hard coded UserDetails for Entered Username.
  - **MyUserDetailsService** (to return hard coded UserDetails for Entered Username)
  - @Secured("ROLE_ADMIN") (to Restrict Access to /Hello Endpoint)
  - Default Login Form (to enter Username and Password)

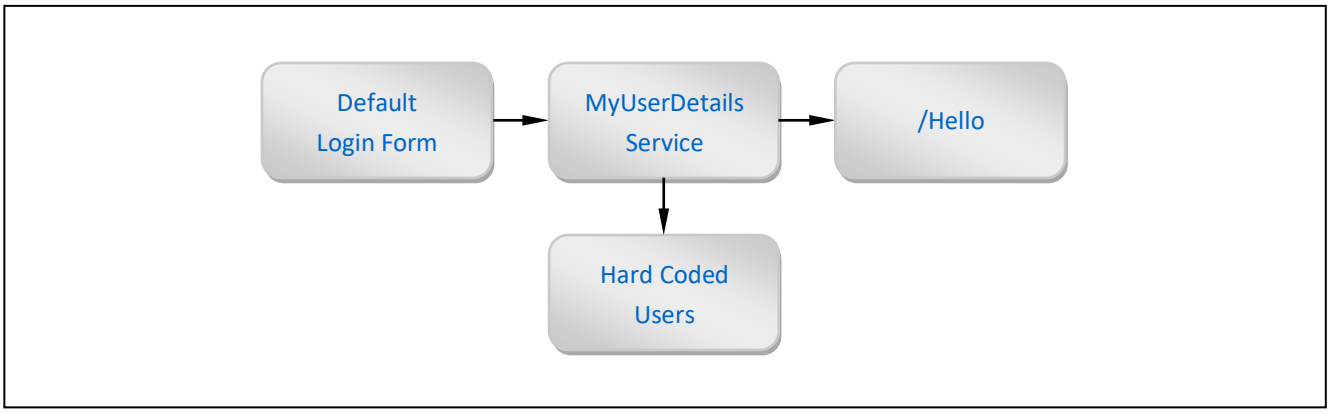
Application Schema *[Results]*



Spring Boot Starters

GROUP	DEPENDENCY	DESCRIPTION
Web	Spring Web	Enables: @RestController, @RequestMapping, Tomcat Server
Security	Spring Security	Enables: @Secured

Authorization Process



Procedure

- Create Project: springboot_security_users_myuserdetailsservice (add Spring Boot Starters from the table)
- Create Package: services (inside main package)
  - Create Class: MyUserDetailsService.java (inside package services)
- Create Package: config (inside main package)
  - Create Class: WebSecurityConfig.java (inside package config)
- Create Package: controllers (inside main package)
  - Create Class: MyController.java (inside package controllers)

MyUserDetailsService.java

```
package com.ivoronline.springboot_security_users_myuserdetailsservice.services;

import org.springframework.security.core.GrantedAuthority;
import org.springframework.security.core.authority.SimpleGrantedAuthority;
import org.springframework.security.core.userdetails.User;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.core.userdetails.UsernameNotFoundException;
import org.springframework.stereotype.Service;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;

@Service
public class MyUserDetailsService implements UserDetailsService {

  @Override
  public UserDetails loadUserByUsername(String enteredUsername) throws UsernameNotFoundException {

    //-----------------------------------------------------------------------------------------------
    //MOCK DB (Hard Coded Users): <username, {"password", "ROLE"}>
    HashMap<String, String[]> accounts = new HashMap<>();
                              accounts.put("myuser" , new String[]{"myuserpassword" , "ROLE_USER" });
                              accounts.put("myadmin", new String[]{"myadminpassword", "ROLE_ADMIN"});
    //-----------------------------------------------------------------------------------------------

    //GET USER/ACCOUNT (From DB)
    String[] account = accounts.get(enteredUsername);

    //CHECK IF USER/ACCOUNT EXISTS
    if (account == null) { throw new UsernameNotFoundException(enteredUsername); }  //Bad credentials

    //GET PASSWORD
    String storedPassword = account[0];

    //GET AUTHORITIES
    List<GrantedAuthority> authorities = new ArrayList<GrantedAuthority>();
                           authorities.add(new SimpleGrantedAuthority(account[1]));

    //CREATE USER DETAILS OBJECT
    UserDetails userDetails = new User(enteredUsername, storedPassword, authorities);

    //RETURN USER
    return userDetails;

  }

}
```

WebSecurityConfig.java

```
package com.ivoronline.springboot_security_users_myuserdetailsservice.config;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.config.annotation.method.configuration.EnableGlobalMethodSecurity;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;
import org.springframework.security.crypto.password.NoOpPasswordEncoder;
import org.springframework.security.crypto.password.PasswordEncoder;

@Configuration
@EnableGlobalMethodSecurity(securedEnabled = true)
public class WebSecurityConfig extends WebSecurityConfigurerAdapter {

  //=========================================================================
  // PASSWORD ENCODER
  //=========================================================================
  @Bean
  PasswordEncoder passwordEncoder() {
    return NoOpPasswordEncoder.getInstance();
  }

  //=========================================================================
  // CONFIGURE
  //=========================================================================
  @Override
  protected void configure(HttpSecurity httpSecurity) throws Exception {
    httpSecurity.authorizeRequests().antMatchers("/Authenticate").permitAll(); //ANONYMOUS ACCESS
    httpSecurity.formLogin();                                                   //DEFAULT LOGIN FORM
  }

}
```

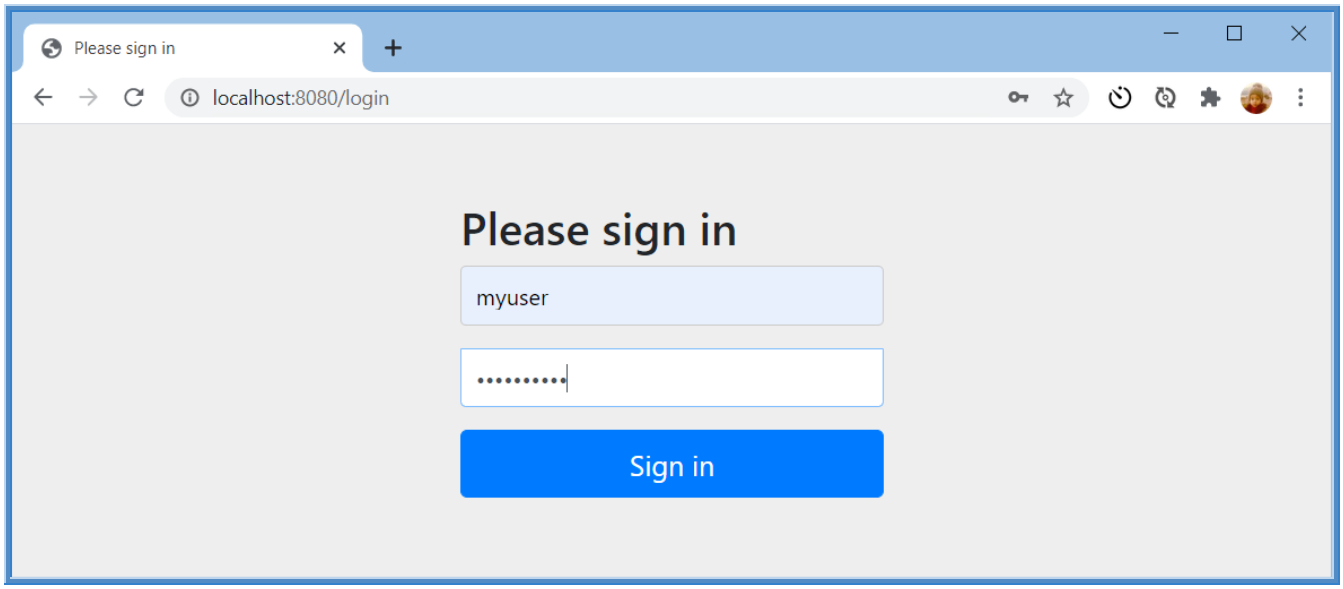
MyController.java

```
package com.ivoronline.springboot_security_users_myuserdetailsservice.controllers;

import org.springframework.security.access.annotation.Secured;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class MyController {

  //=============================================================================
  // HELLO (Secured Resource)
  //=============================================================================
  @Secured("ROLE_USER")
  @RequestMapping("/Hello")
  public String hello() {
    return "Hello from Controller";
  }

}
```

Results

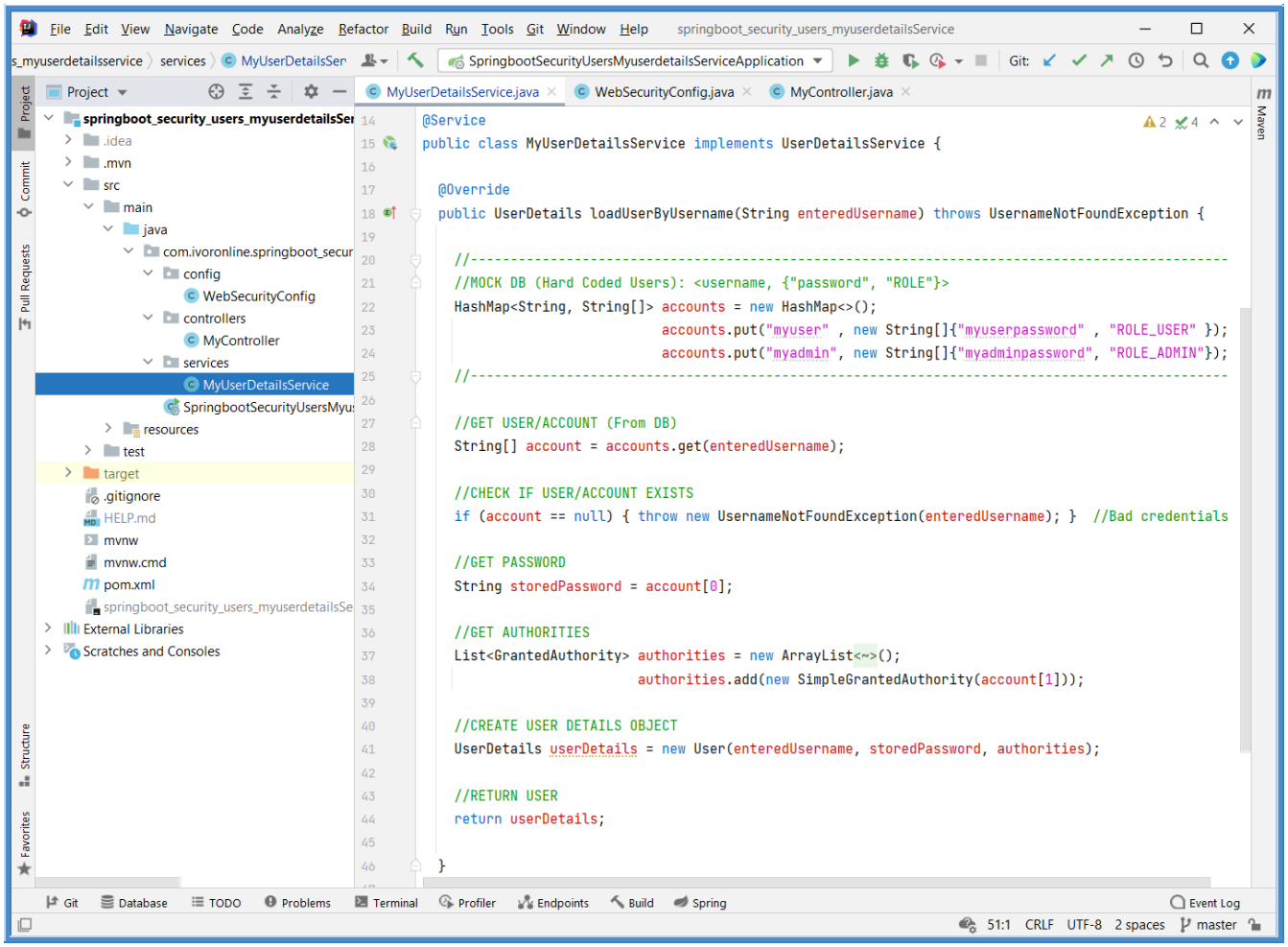
- http://localhost:8080/Hello
- You get redirected to http://localhost:8080/login
  - Username: **myuser**
  - Password: **myuserpassword**
  - Sign in
- You get redirected back to http://localhost:8080/Hello

http://localhost:8080/login - myuser - myuserpassword



Redirects to http://localhost:8080/Hello

Application Structure



pom.xml

```
<dependencies>

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

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

</dependencies>
```

2.2 Add Authorities to Endpoints

Info

- Following tutorials show a different ways of assigning Authorities/Roles to Endpoints.
- Since Authorities/Roles are also assigned to Users this defines which Users have access to which Endpoints.

Content

authorizeRequests()	(assign Roles & Authorities	to multiple	Endpoints)
@Secured	(assign Roles	to single	Endpoint)
@PreAuthorize	(assign Roles & Authorities	to single	Endpoint)
@PreAuthorize - Custom Methods	(call Custom Method that returns Boolean to control access to Endpoint)		

authorizeRequests()

- authorizeRequests() Method allows you to assign Roles & Authorities to selected Endpoints
  - Ant Matchers are used to **select** multiple Endpoints (to which Roles & Authorities should be added)
  - followed by Methods that **assign Roles & Authorities** to selected Endpoints

SecurityConfig.java

```
httpSecurity.authorizeRequests()
  .antMatchers("/endPoint1").denyAll()
  .antMatchers("/endPoint2", "/endPoint3").permitAll()
  .antMatchers("/*", "/**", "/end*").permitAll()
```

- Security Expressions - API allows you to define Roles and Authorities through API Expression.

SecurityConfig.java

```
httpSecurity.authorizeRequests()
  .antMatchers("/endPoint1").denyAll()                    //No access       (even after log in)
  .antMatchers("/endPoint2").permitAll()                  //All have access (anonymous access without Login)
  .antMatchers("/endPoint3").hasRole("ADMIN")             //Only ADMIN ROLE      can access (after log in)
  .antMatchers("/endPoint4").hasAnyRole("ADMIN", "USER"); //Only ADMIN/USER ROLE can access (after log in)
```

@Secured

- @Secured Annotation allows you to assign Roles to single Endpoint.

SecurityConfig.java *(enables @Secured Annotation)*

```
@EnableGlobalMethodSecurity(securedEnabled = true)
```

MyController.java

```
@Secured( "ROLE_ADMIN")
@Secured({"ROLE_ADMIN", "ROLE_USER"})
```

@PreAuthorize

- @PreAuthorize Annotation allows you to assign Roles & Authorities to single Endpoint.

WebSecurityConfig.java *(enables @PreAuthorize Annotation)*

```
@EnableGlobalMethodSecurity(prePostEnabled = true)
```

MyController.java

```
@PreAuthorize("hasRole   ('ADMIN')")
@PreAuthorize("hasAnyRole('ADMIN', 'USER')")
```

@PreAuthorize - Custom Methods

- `@PreAuthorize` Annotation can also call additional Custom Method that returns Boolean to control access to Endpoint.

Roles vs Authorities

- Roles & Authorities are technically the same - they can be used to achieve the same purpose in exactly the same way.
- This means that wherever you are using Roles you could replace them with Authorities and vice versa.
- Confusion between the two arises for two reasons
  - they are referenced differently
  - they are intended to be used for two different security design patterns

Differences in referencing

- Roles & Authorities have their own sets of Methods `hasRole()` vs `hasAuthority()`
- Role must have prefix ROLE_ `ROLE_ADMIN`
- Role is referenced by omitting prefix ROLE_ `hasRole("ADMIN")`
- Authority is referenced by using its full name `hasAuthority("author.read")`

Differences in design pattern

- Although technically the same, Roles & Authorities are conceptually/logically different.
- As such it makes logically more sense to use one over the other depending on the chosen security design pattern.
- The main difference between these two security design patterns are that
  - single Role is assigned to multiple Endpoints `@Secured("ROLE_ADMIN")`
  - each Endpoint gets unique Authority `@Secured("author.read")`

Roles

- You can think of Roles as simple out of the box solution for implementing security (that doesn't require any custom code).
- Using security design pattern with Roles is simpler of the two
  - **assign the same Role to multiple Endpoints**
  - assign that Role to a User
  - this way you have defined to which Endpoints User has access to
  - the whole configuration is contained inside the Controller using simple Annotations (no need for custom Entities)

Authorities

- Using security design pattern with Authorities is more complex of the two
  - **assign unique Authority to each Endpoint**
  - create custom Entity Profile which will be used to group Authorities
  - inside the Database assign multiple Authorities to each Profile
  - assign Profile to Account
- When you get Account from DB
  - you get its Profile
  - then you get Authorities related to that Profile
  - then you create User with all of these Authorities
  - this way you have defined to which Endpoints Account/User has access to
- Alternatively instead of using DB you could define Profiles and related Authorities through some Configuration File/Class.
- As you can see this is far more complicated approach compared to using Roles.
  In this approach security configuration is no longer encapsulated in Controller (which User has access to which Endpoint).
  Instead configuration is moved to DB through Profile Entity which groups Authorities (kind of replacing Spring Role).

Usage

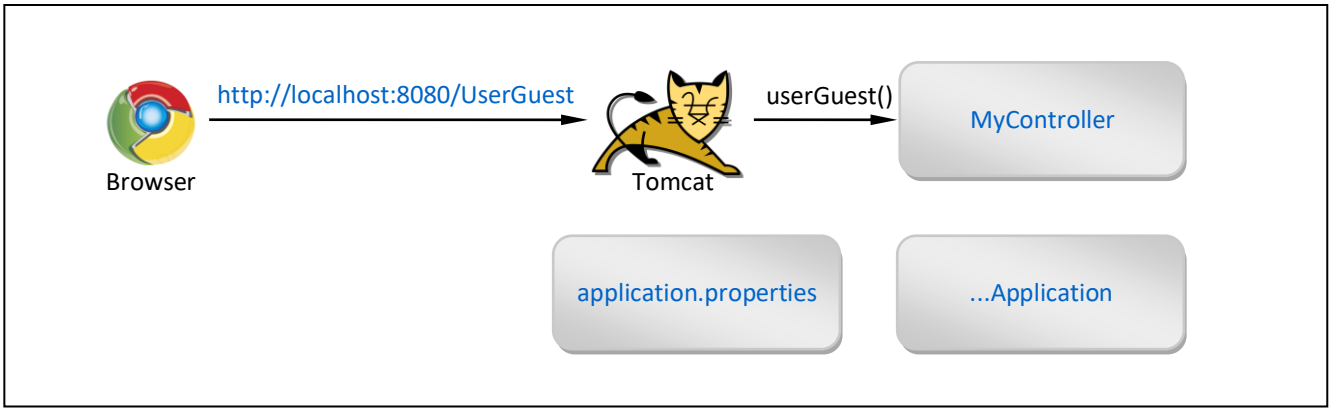
- In your application you should use either Roles or Authorities - but not both in the same application.
- Technically you could use them both but that is likely to lead to a confusion.
- That is because you will have security configuration in two different places - Controller and DB.

2.2.1 @Secured - Roles

Info **[G]**

- This tutorial shows how to use `@Secured` to add multiple Roles to single Endpoint.
- We will use
  - **application.properties** (to define Users with Roles)
  - `@Secured({"USER", "GUEST"})` (to assign multiple Roles to single Endpoint)
  - **Default** Login Form (to enter Credentials)

Application Schema *[Results]*



Spring Boot Starters

GROUP	DEPENDENCY	DESCRIPTION
Web	Spring Web	Enables: @RestController, @RequestMapping, Tomcat Server
Security	Spring Security	Enables: @Secured

SpringbootSecuritySecuredApplication.java

```
@EnableGlobalMethodSecurity(securedEnabled = true)
```

MyController.java

```
@Secured("ADMIN")
@Secured({"USER", "GUEST"})
```

Procedure

- Create Project: springboot_security_expression_secured (add Spring Boot Starters from the table)
- Edit File: application.properties (specify Username, Password, Role)
- Edit Class: SpringbootSecuritySecuredApplication.java (@EnableGlobalMethodSecurity)
- Create Package: controllers (inside main package)
  - Create Class: MyController.java (inside controllers package)

application.properties

```
# SECURITY
spring.security.user.name     = myuser
spring.security.user.password = myuserpassword
spring.security.user.roles    = USER, GUEST
```

SpringbootSecuritySecuredApplication.java

```
package com.ivoronline.springboot_security_secured;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.security.config.annotation.method.configuration.EnableGlobalMethodSecurity;

@SpringBootApplication
@EnableGlobalMethodSecurity(securedEnabled = true)
public class SpringbootSecuritySecuredApplication {

  public static void main(String[] args) {
    SpringApplication.run(SpringbootSecuritySecuredApplication.class, args);
  }

}
```

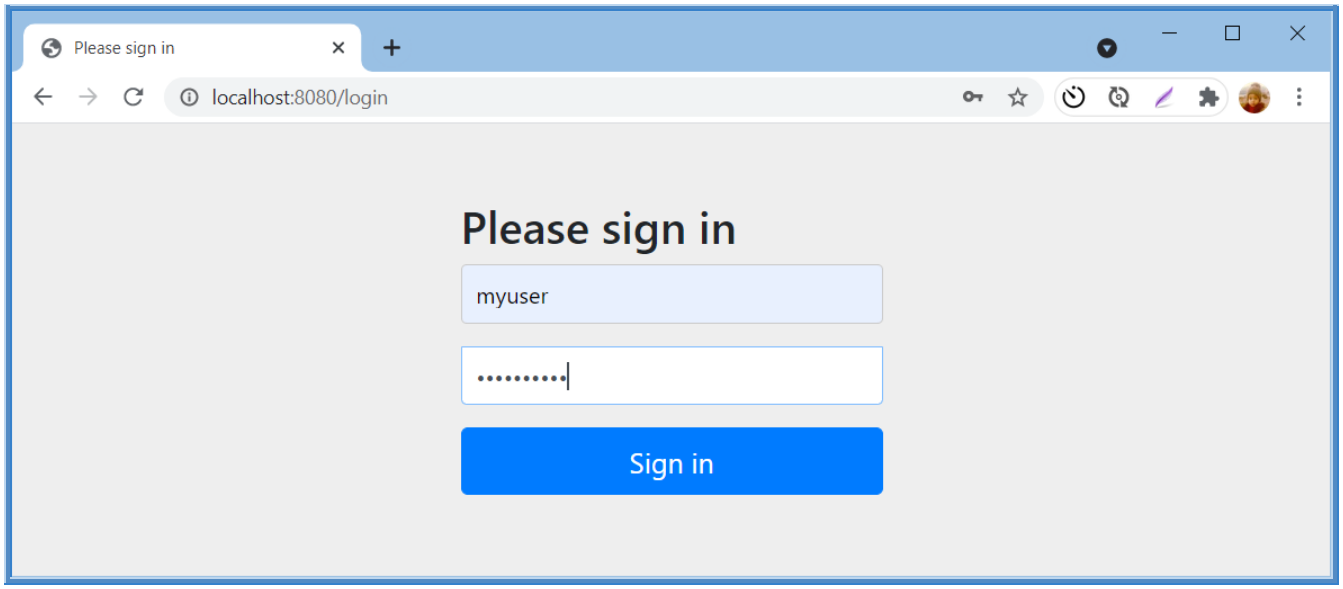
MyController.java

```
package com.ivoeonline.springboot_security_preauthorize.controllers;

import org.springframework.security.access.prepost.PreAuthorize;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class MyController {

  @PreAuthorize("hasRole('ADMIN')")
  @RequestMapping("/Admin")
  public String admin() {
    return "Hello from Admin";
  }

  @PreAuthorize("hasAnyRole('ADMIN', 'USER')")
  @RequestMapping("/UserGuest")
  public String userGuest() {
    return "Hello from UserGuest";
  }

}
```

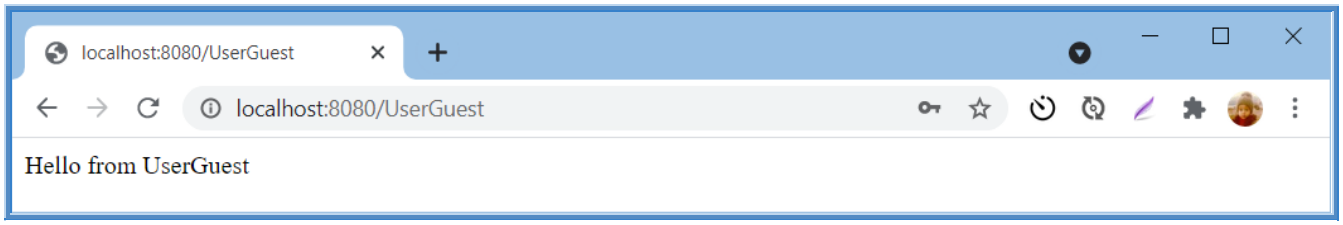
Results

- http://localhost:8080/UserGuest
- You get redirected to http://localhost:8080/login
  - Username: **myuser**
  - Password: **myuserpassword**
  - Sign in
- You get redirected back to http://localhost:8080/UserGuest (allowed access)
- http://localhost:8080/Admin (denied access)

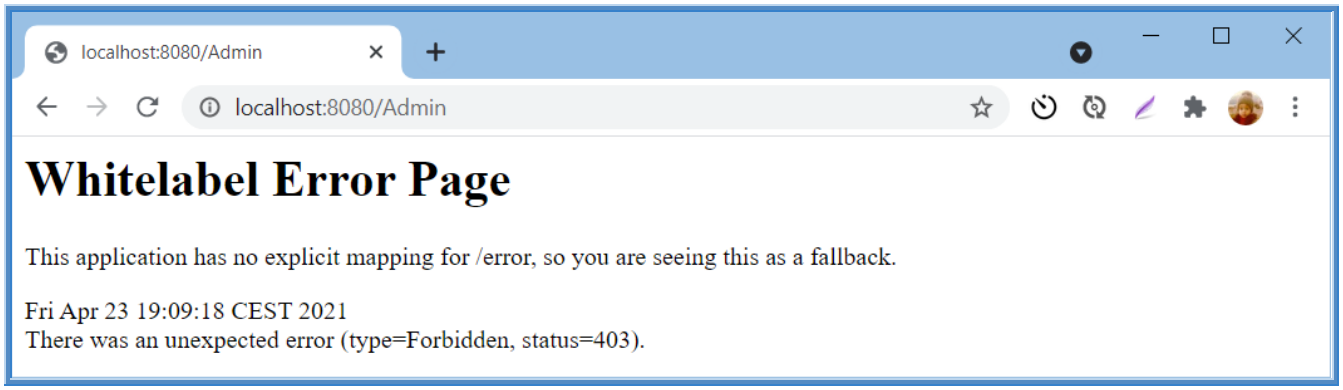
http://localhost:8080/login



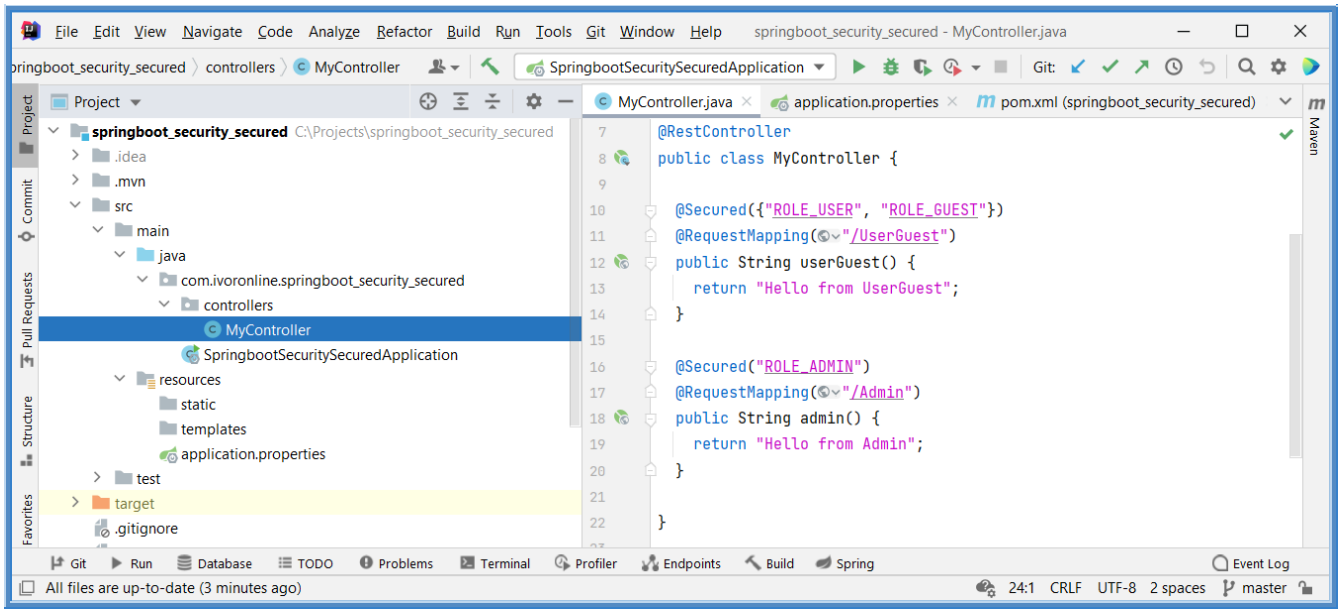
http://localhost:8080/UserGuest



http://localhost:8080/Admin



Application Structure



pom.xml

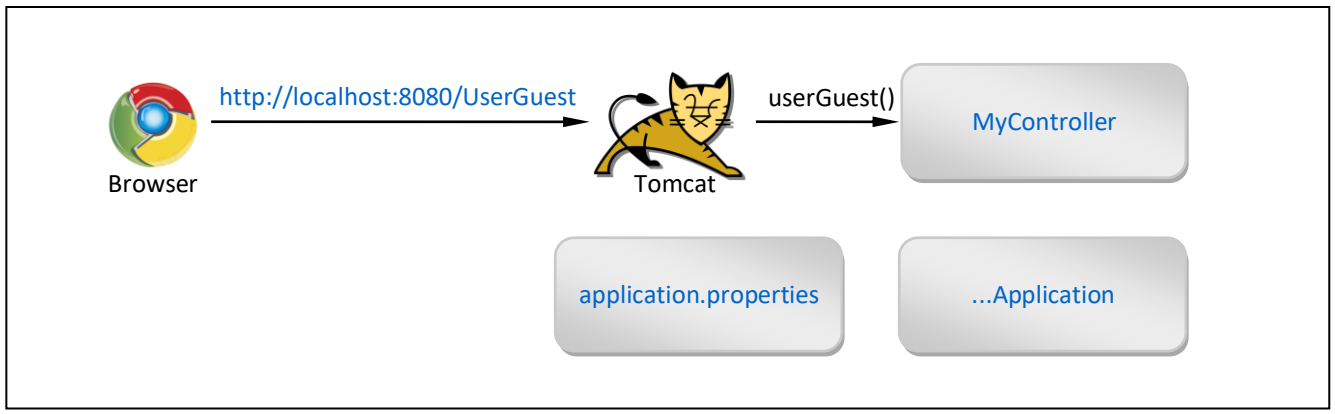
```
<dependencies>

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

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

</dependencies>
```

2.2.2 @PreAuthorize - Roles & Authorities

Info [G]

- This tutorial shows how to use `@PreAuthorize` to add multiple Authorities to single Endpoint.
- We will use
  - **application.properties** (to define Users with Roles)
  - `@PreAuthorize("hasAnyRole('ADMIN', 'USER')")` (to assign multiple Roles & Authorities to single Endpoint)
  - **Default** Login Form (to enter Credentials)

Application Schema *[Results]*



Spring Boot Starters

GROUP	DEPENDENCY	DESCRIPTION
Web	Spring Web	Enables: @RestController, @RequestMapping, Tomcat Server
Security	Spring Security	Enables: @Secured

SecurityConfig.java

```
@EnableGlobalMethodSecurity(prePostEnabled = true)
```

MyController.java

```
@PreAuthorize("hasRole        ('ADMIN')                 ")
@PreAuthorize("hasAnyRole     ('ADMIN', 'USER')         ")
@PreAuthorize("hasAuthority   ('ROLE_ADMIN')            ")
@PreAuthorize("hasAnyAuthority('ROLE_ADMIN', 'ROLE_USER')")
```

Procedure

- Create Project: springboot_security_expression_secured (add Spring Boot Starters from the table)
- Edit File: application.properties (specify Username, Password, Role)
- Edit Class: SpringbootSecurityPreauthorizeApplication.java (inside config package)
- Create Package: controllers (inside main package)
  - Create Class: MyController.java (inside controllers package)

application.properties

```
# SECURITY
spring.security.user.name     = myuser
spring.security.user.password = myuserpassword
spring.security.user.roles    = USER, GUEST
```

SpringbootSecurityPreauthorizeApplication.java

```
package com.ivoeonline.springboot_security_preauthorize;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.security.config.annotation.method.configuration.EnableGlobalMethodSecurity;

@SpringBootApplication
@EnableGlobalMethodSecurity(prePostEnabled = true)
public class SpringbootSecurityPreauthorizeApplication {

  public static void main(String[] args) {
    SpringApplication.run(SpringbootSecurityPreauthorizeApplication.class, args);
  }

}
```

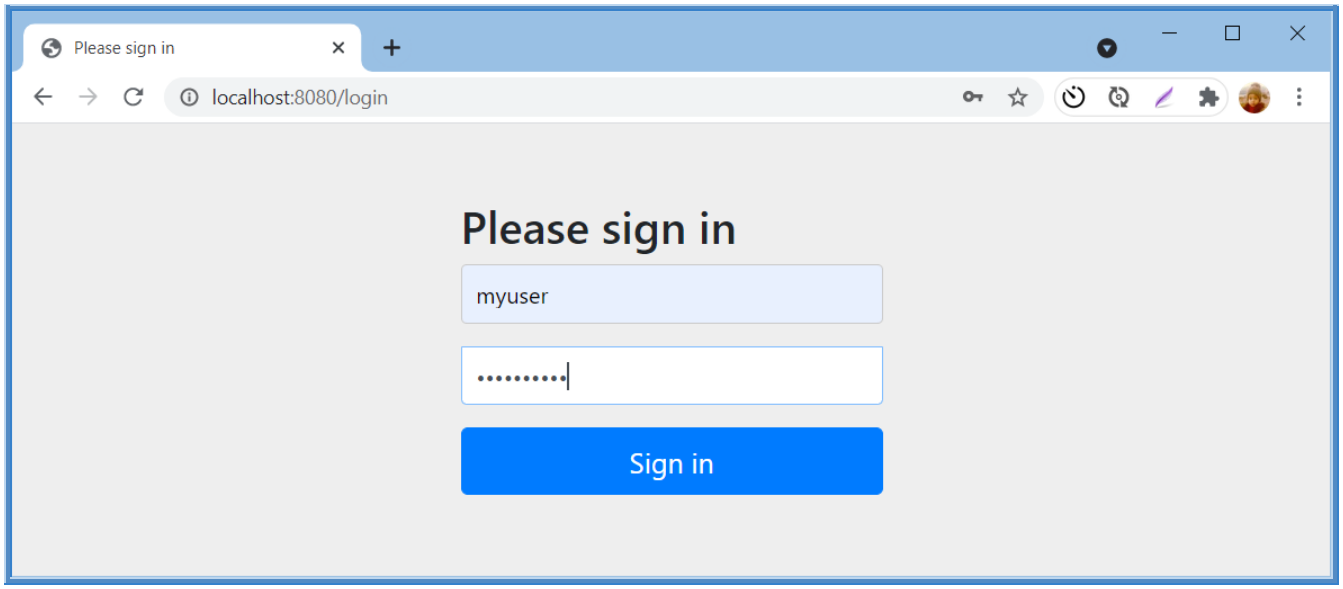
MyController.java

```
package com.ivoeonline.springboot_security_preauthorize.controllers;

import org.springframework.security.access.prepost.PreAuthorize;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class MyController {

  @PreAuthorize("hasRole('ADMIN')")
  @RequestMapping("/Admin")
  public String admin() {
    return "Hello from Admin";
  }

  @PreAuthorize("hasAnyRole('ADMIN', 'USER')")
  @RequestMapping("/UserGuest")
  public String userGuest() {
    return "Hello from UserGuest";
  }

}
```

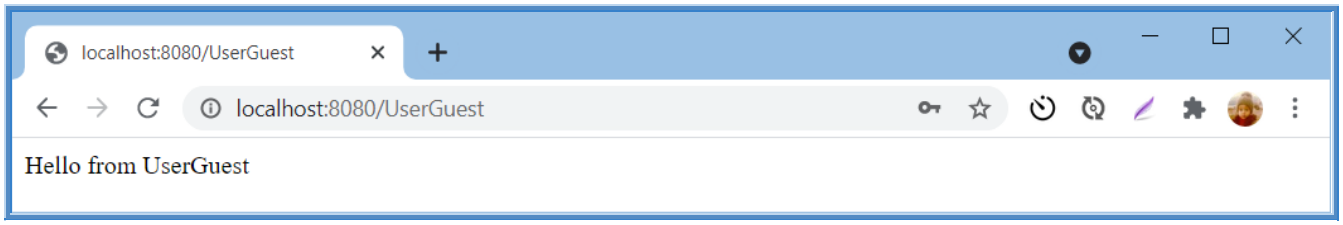
Results

- http://localhost:8080/UserGuest
- You get redirected to http://localhost:8080/login
  - Username: **myuser**
  - Password: **myuserpassword**
  - Sign in
- You get redirected back to http://localhost:8080/UserGuest (allowed access)
- http://localhost:8080/Admin (denied access)

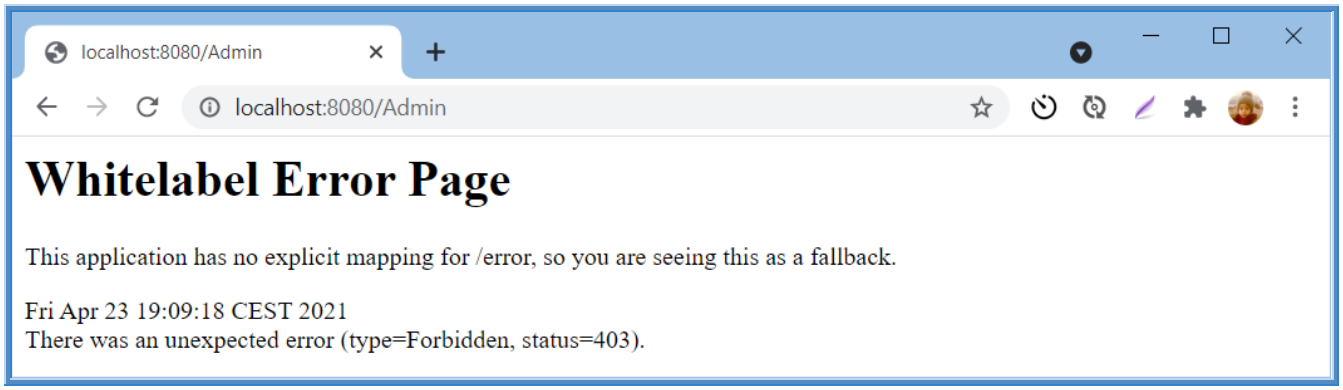
http://localhost:8080/login



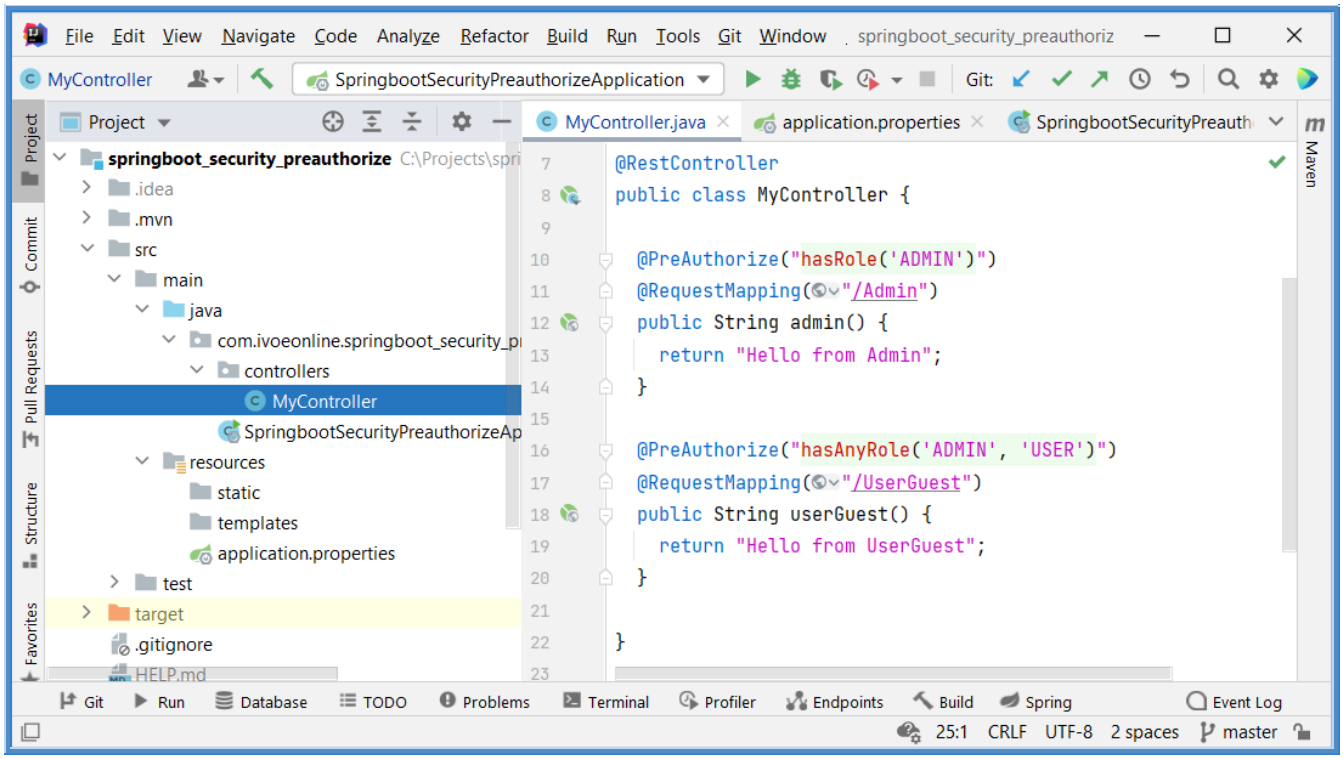
http://localhost:8080/UserGuest



http://localhost:8080/Admin



Application Structure



pom.xml

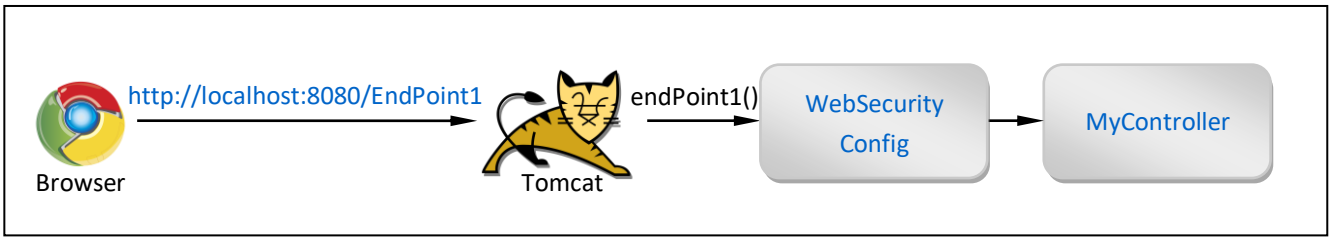
```
<dependencies>

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

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

</dependencies>
```

2.2.3 antMatchers() - Select Endpoints

Info [G]

- When you add Spring Boot Starter **Security** every URL becomes unavailable unless you Log In.
- This tutorial shows how to loosen up this restriction by allowing access to certain URLs without having to Log In.
- We will do this by creating a Class SecurityConfig which extends and Overrides Method configure().

Application Schema *[Results]*



Spring Boot Starters

GROUP	DEPENDENCY	DESCRIPTION
Web	Spring Web	Enables: @RestController, @RequestMapping, Tomcat Server
Security	Spring Security	Enables Spring Security

Procedure

- Create Project: springboot_security_expressions_api (add Spring Boot Starters from the table)
- Create Package: controllers (inside main package)
  - Create Class: MyController.java (inside controllers package)
- Create Package: config (inside main package)
  - Create Class: WebSecurityConfig.java (inside config package)

MyController.java

```
package com.ivoronline.springboot_security_urlpatternmatching.controllers;

import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.ResponseBody;

@Controller
public class MyController {
  @ResponseBody @RequestMapping("/hello")              public String sayHello() { return "hello";     }
  @ResponseBody @RequestMapping("/endPoint1")          public String ep1()      { return "endPoint1"; }
  @ResponseBody @RequestMapping("/endPoint2")          public String ep2()      { return "endPoint2"; }
  @ResponseBody @RequestMapping("/endPoint3")          public String ep3()      { return "endPoint3"; }
  @ResponseBody @RequestMapping("/endPoint4")          public String ep4()      { return "endPoint4"; }
  @ResponseBody @RequestMapping("/sublevel/endPoint5") public String ep5()      { return "endPoint5"; }
}
```

WebSecurityConfig.java

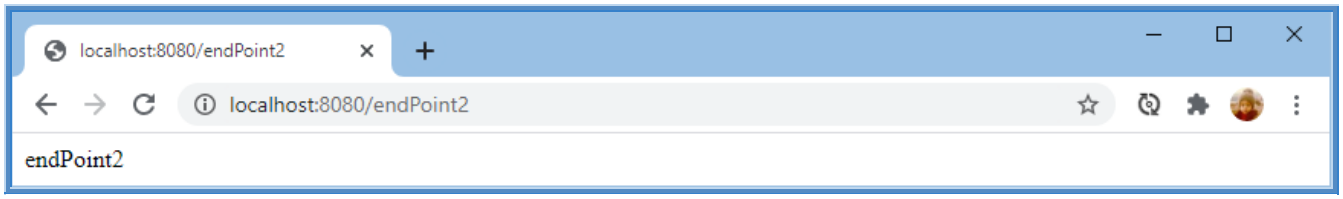
```
package com.ivoronline.springboot_security_urlpatternmatching.config;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.config.annotation.authentication.builders.AuthenticationManagerBuilder;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;
import org.springframework.security.crypto.password.NoOpPasswordEncoder;
import org.springframework.security.crypto.password.PasswordEncoder;

@Configuration
public class WebSecurityConfig extends WebSecurityConfigurerAdapter {

  //=====================================================================
  // CONFIGURE (Auth...)
  //=====================================================================
  @Override
  protected void configure(AuthenticationManagerBuilder auth) throws Exception {
    auth.inMemoryAuthentication().withUser("myuser" ).password("myuserpassword" ).roles("USER" );
    auth.inMemoryAuthentication().withUser("myadmin").password("myadminpassword").roles("ADMIN");
  }

  //=====================================================================
  // PASSWORD ENCODER
  //=====================================================================
  @Bean
  PasswordEncoder passwordEncoder() {
    return NoOpPasswordEncoder.getInstance();
  }

  //=====================================================================
  // CONFIGURE
  //=====================================================================
  @Override
  protected void configure(HttpSecurity httpSecurity) throws Exception {

    //SELECT SPECIFIC ENDPOINTS
    httpSecurity.authorizeRequests()
      .antMatchers("/endPoint1")              .hasRole("USER")  //Select single  Endpoint
      .antMatchers("/endPoint2", "/endPoint3").hasRole("USER"); //Select multiple Endpoints from the list

    //SELECT ENDPOINTS - USING ASTERISK '*'
    httpSecurity.authorizeRequests()
      .antMatchers("/*"                 ).hasRole("USER")  //Select Endpoints-this level
      .antMatchers("/cars/*"            ).hasRole("USER")  //Select Endpoints-this level
      .antMatchers("/cars/end*"         ).hasRole("USER")  //Select Endpoints-this level-start with /end
      .antMatchers("/cars/*", "/planes*" ).hasRole("USER"); //Select Endpoints that match listed URL Patterns

    //SELECT ENDPOINTS - USING DOUBLE ASTERISK '**'
    httpSecurity.authorizeRequests()
      .antMatchers("/**"                ).hasRole("USER")  //Select Endpoints-any level
      .antMatchers("/end**"             ).hasRole("USER")  //Select Endpoints-any level-start with /end
      .antMatchers("/cars/**"           ).hasRole("USER")  //Select Endpoints-any level-start with /cars/
      .antMatchers("/cars/end**"        ).hasRole("USER")  //Select Endpoints-any level-start with this
      .antMatchers("/cars/*", "/bikes/**").hasRole("USER"); //Select Endpoints that match listed URL Patterns

    //AUTHENTICATE WITH DEFAULT LOGIN FORM
    httpSecurity.formLogin(); //Without this there is no Login Form to Authenticate

  }

}
```

Results

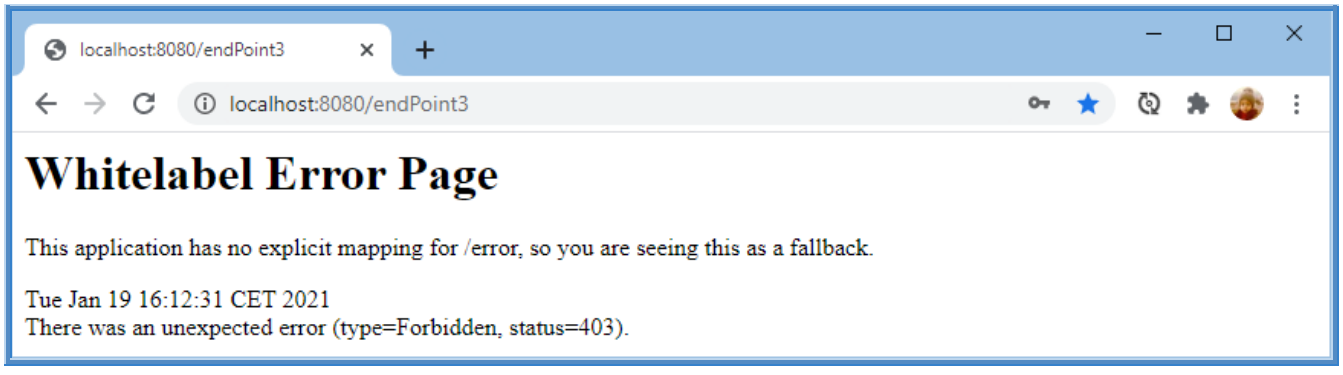
http://localhost:8080/login - myuser - myuserpassword

http://localhost:8080/endPoint1 — *denyAll() => deny **ALL** after Login*

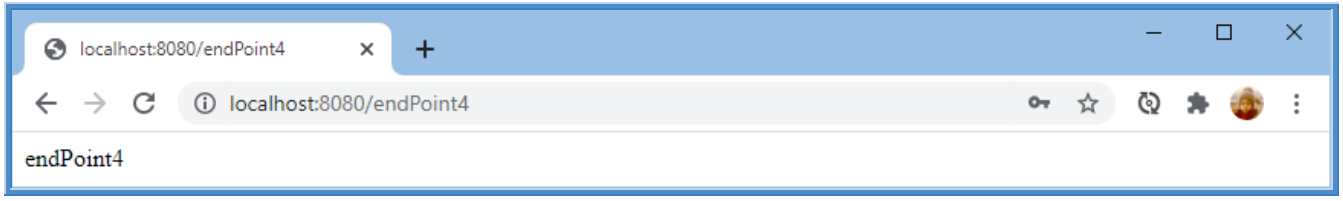
http://localhost:8080/endPoint2 — *permitAll() => allow **ALL** without Login*



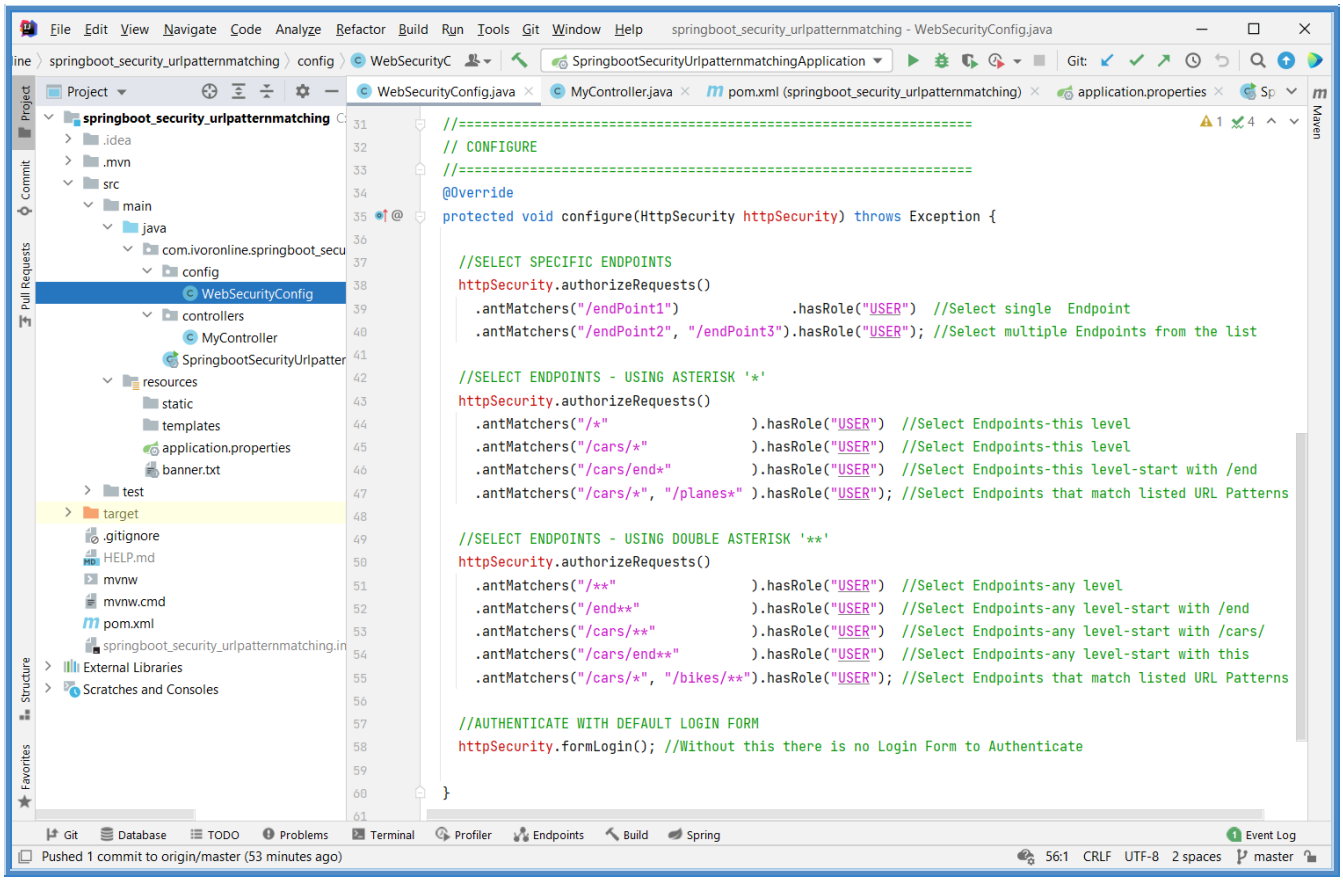
http://localhost:8080/endPoint3 — *hasRole("ADMIN") => deny **USER** after Login*



http://localhost:8080/endPoint4 — *hasAnyRole("ADMIN", "USER") => allow **USER** after Login*



Application Structure



pom.xml

```
<dependencies>

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

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

</dependencies>
```

2.3 Read Credentials

Info

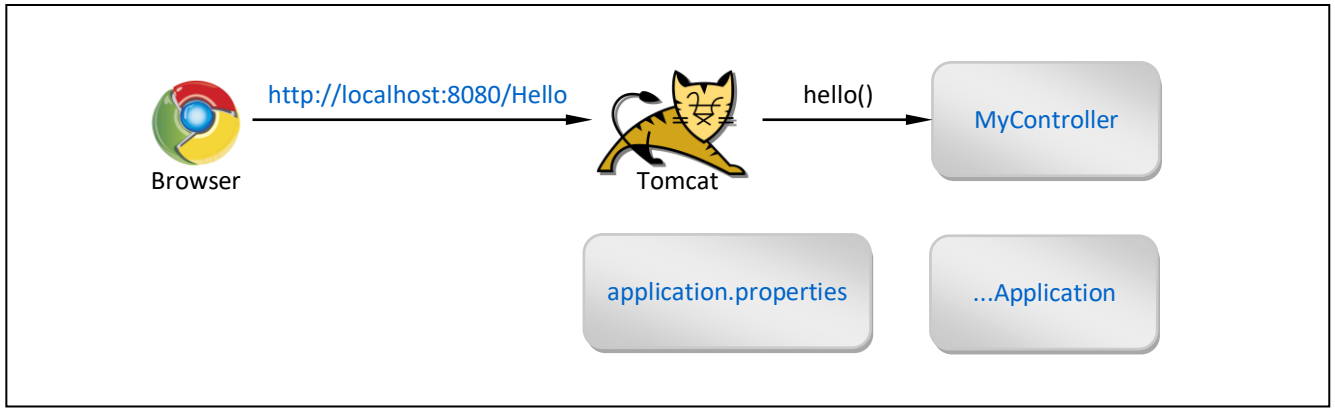
- Following tutorials show different ways of reading credentials from HTTP Request.
- Tutorials are organized in two groups
  - tutorials which show how Spring can **Automatically** get Credentials from HTTP Request
  - tutorials which show how you can **Manually** get Credentials from HTTP Request
- Automatic and Manual - Controller tutorials use
  - authenticationManagerBean() to instantiate AuthenticationManager
  - userDetailsService() to define Users
- Automatic & Manual - Controller tutorials use **Session** to allow User to access Restricted Resources after Authentication
  Manual - Filter tutorials **do not use Session** which means that Credentials need to be provided with every HTTP Request.

2.3.1 Automatic - Login Form - Default

Info [G]

- This tutorial shows how to **Automatically get Credentials** from **Default** Login Form.
- We will use
  - application.properties (to define User with Roles)
  - @Secured("ROLE_ADMIN") (to add Authorities to Endpoint /Hello)
  - **Default Login Form** (to read Credentials)

Application Schema *[Results]*



Spring Boot Starters

GROUP	DEPENDENCY	DESCRIPTION
Web	Spring Web	Enables: @RestController, @RequestMapping, Tomcat Server
Security	Spring Security	Enables: @Secured

http://localhost:8080/login *(default Login Form)*

Procedure

- Create Project: springboot_security_loginform_default (add Spring Boot Starters from table)
- Edit File: application.properties (add Role, User, Password)
- Edit Class: SpringbootSecurityLoginformDefaultApplication.java (@EnableGlobalMethodSecurity)
- Create Package: controllers (inside main package)
  - Create Class: MyController.java (inside controllers package)

application.properties

```
# SECURITY
spring.security.user.name     = myuser
spring.security.user.password = myuserpassword
spring.security.user.roles    = USER, ADMIN
```

SpringbootSecurityLoginformDefaultApplication.java

```
package com.ivoronline.springboot_security_loginform_default;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.security.config.annotation.method.configuration.EnableGlobalMethodSecurity;

@SpringBootApplication
@EnableGlobalMethodSecurity(securedEnabled = true)
public class SpringbootSecurityLoginformDefaultApplication {

  public static void main(String[] args) {
    SpringApplication.run(SpringbootSecurityLoginformDefaultApplication.class, args);
  }

}
```

MyController.java

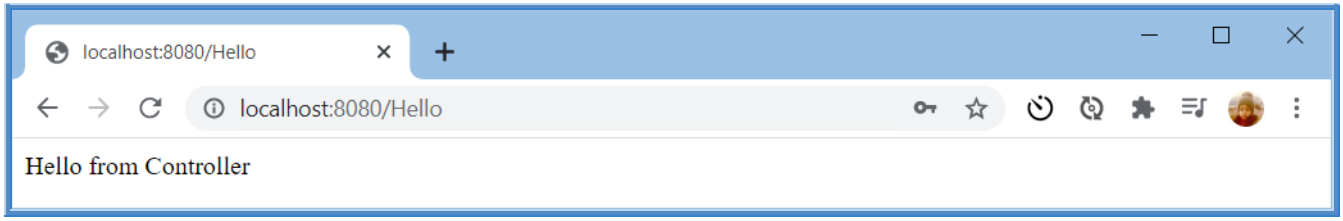
```
package com.ivoronline.springboot_security_loginform_default.controllers;

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

@RestController
public class MyController {

  //=============================================================================
  // HELLO (Secured Resource)
  //=============================================================================
  @Secured("ROLE_USER")
  @RequestMapping("/Hello")
  public String hello() {
    return "Hello from Controller";
  }

}
```

Results

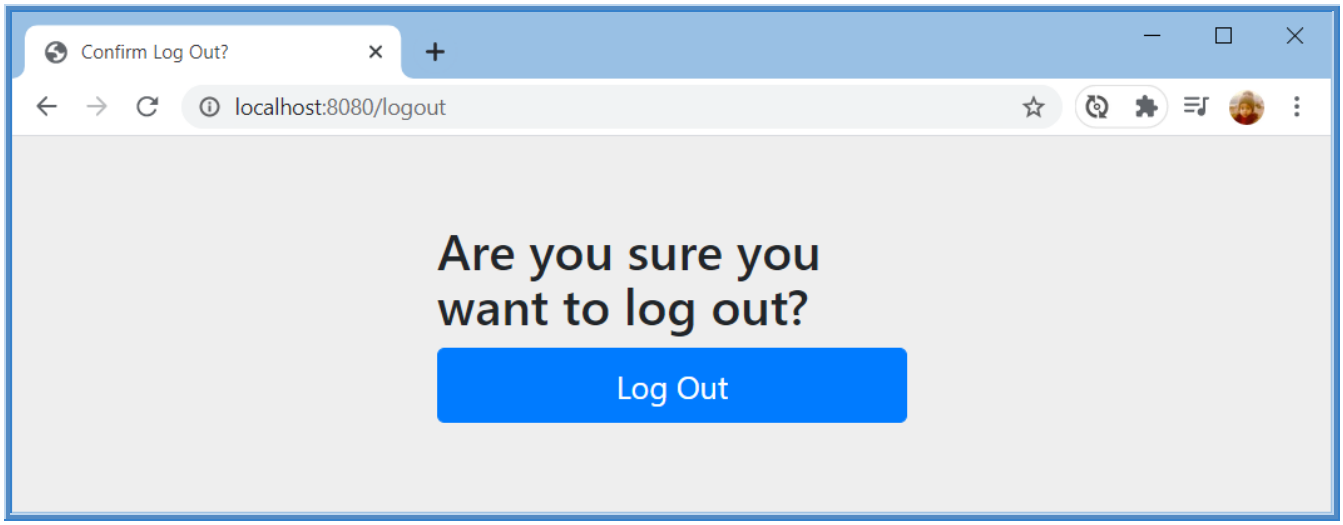
- http://localhost:8080/Hello
- You get redirected to http://localhost:8080/login
  - Username: **myuser**
  - Password: **myuserpassword**
  - Sign in
- You get redirected back to http://localhost:8080/Hello
- Logout Form http://localhost:8080/logout
  - Log Out

http://localhost:8080/login

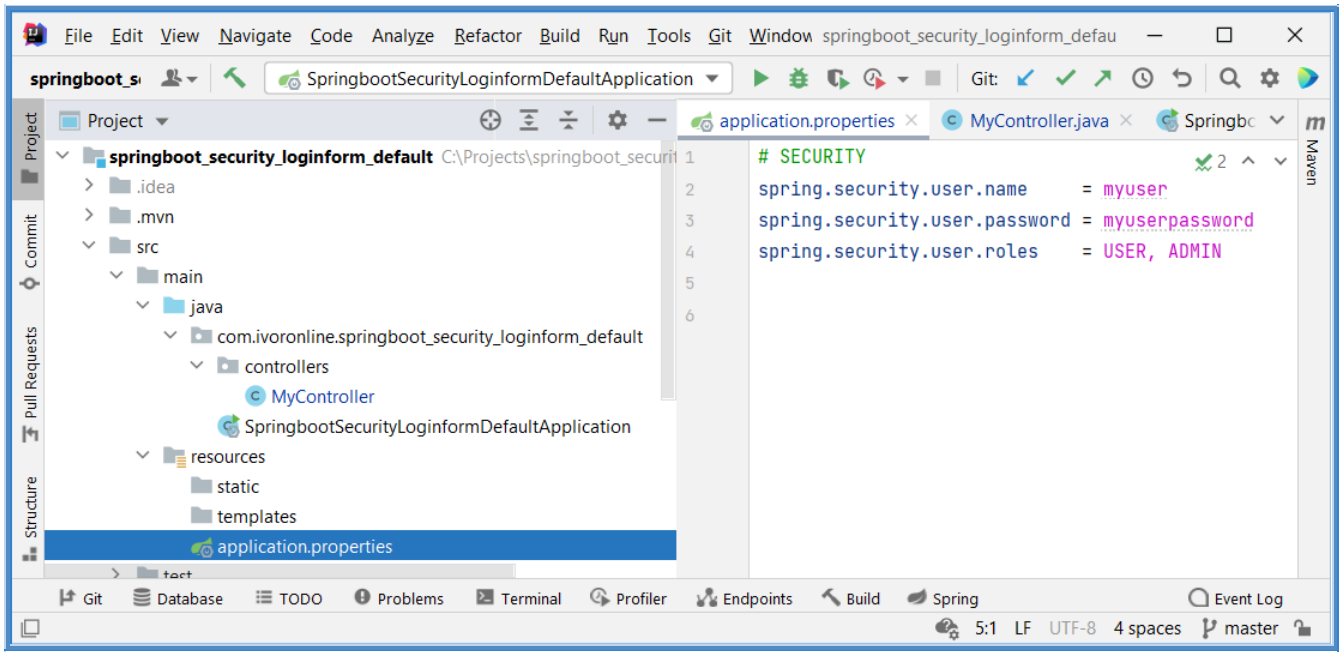
Redirects to http://localhost:8080/Hello



http://localhost:8080/logout



Application Structure



pom.xml

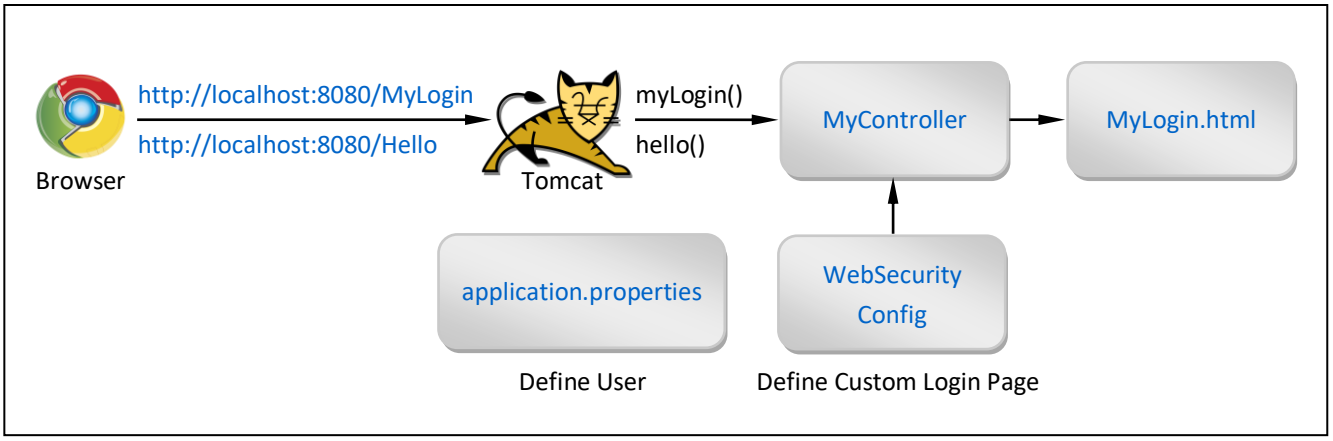
```
<dependencies>

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

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

</dependencies>
```

2.3.2 Automatic - Login Form - Custom

Info [G]

- This tutorial shows how to use **Automatically get Credentials** from **Custom** Login Form.
- We will use
  - application.properties (to define User with Roles)
  - **Custom Login Form** MyLogin.html (to read Credentials, Custom Login Form is specified in WebSecurityConfig)
  - @Secured("ROLE_ADMIN") (to add Authorities to Endpoint /Hello)

Application Schema *[Results]*



Spring Boot Starters

GROUP	DEPENDENCY	DESCRIPTION
Web	Spring Web	Enables: @Controller, @RequestMapping, @ResponseBody, Tomcat Server
Security	Spring Security	Enables: @Secured
Template Engines	Thyemeleaf	Enables Controller to return reference to HTML Page MyLogin.html

Procedure

- Create Project: springboot_security_loginform_custom (add Spring Boot Starters from the table)
- Edit File: application.properties (add Role, User, Password)
- Create Package: controllers (inside main package)
  - Create Class: MyController.java (inside controllers package)
- Create Package: config (inside main package)
  - Create Class: WebSecurityConfig.java (inside config package)
- Create HTML File: MyLogin.html (inside directory resources/templates)

application.properties

```
# SECURITY
spring.security.user.name     = myuser
spring.security.user.password = myuserpassword
spring.security.user.roles    = USER, ADMIN
```

MyController.java

```
package com.ivoronline.springboot_security_loginform_custom.controllers;

import org.springframework.security.access.annotation.Secured;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.ResponseBody;

@Controller
public class MyController {

  //============================================================================
  // MY LOGIN
  //============================================================================
  @RequestMapping("/MyLogin")
  public String myLogin() {
    return "MyLogin";
  }

  //============================================================================
  // HELLO
  //============================================================================
  @ResponseBody
  @Secured("ROLE_USER")
  @RequestMapping("/Hello")
  public String hello() {
    return "Hello from Controller";
  }

}
```

WebSecurityConfig.java

```
package com.ivoronline.springboot_security_loginform_custom.config;

import org.springframework.context.annotation.Configuration;
import org.springframework.security.config.annotation.method.configuration.EnableGlobalMethodSecurity;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;

@Configuration
@EnableGlobalMethodSecurity(securedEnabled = true)
public class WebSecurityConfig extends WebSecurityConfigurerAdapter {

  @Override
  protected void configure(HttpSecurity httpSecurity) throws Exception {

    //CUSTOM LOGIN FORM
    httpSecurity.formLogin()
      .loginPage("/MyLogin")         //Custom  HTML Page for entering   Credentials
      .loginProcessingUrl("/login"); //Default HTML Page for processing Credentials

    //DISABLE CSRF
    httpSecurity.csrf().disable();

  }

}
```

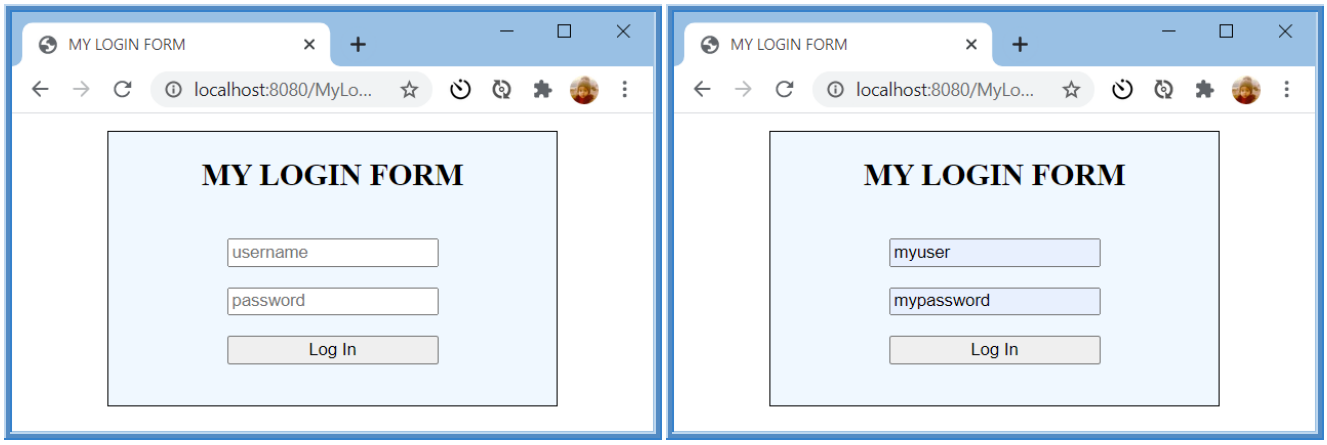
MyLogin.html

```
<title> MY LOGIN </title>

<style type="text/css">
    div { display:flex; flex-direction:column; align-items:center; border: solid 1pt; margin: 10pt 50pt;
background-color: aliceblue }
</style>

<div>
  <h2> MY LOGIN </h2>
  <form method="POST" action="/MyLogin">
    <p> <input type="text"   name="username"  placeholder="username"            /> </p>
    <p> <input type="text"   name="password"  placeholder="password"            /> </p>
    <p> <input type="submit" name="addAuthor" value="Log In" style="width:100%" /> </p>
  </form>
</div>
```

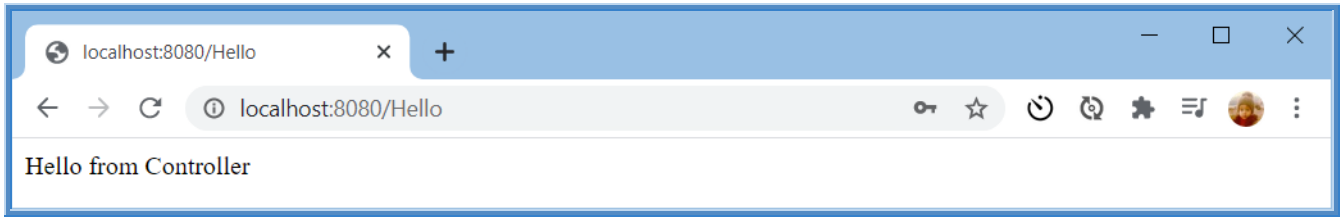
Results

- http://localhost:8080/Hello
- You get redirected to http://localhost:8080/MyLogin
  - Username: **myuser**
  - Password: **myuserpassword**
  - Log in
- You get redirected back to http://localhost:8080/Hello

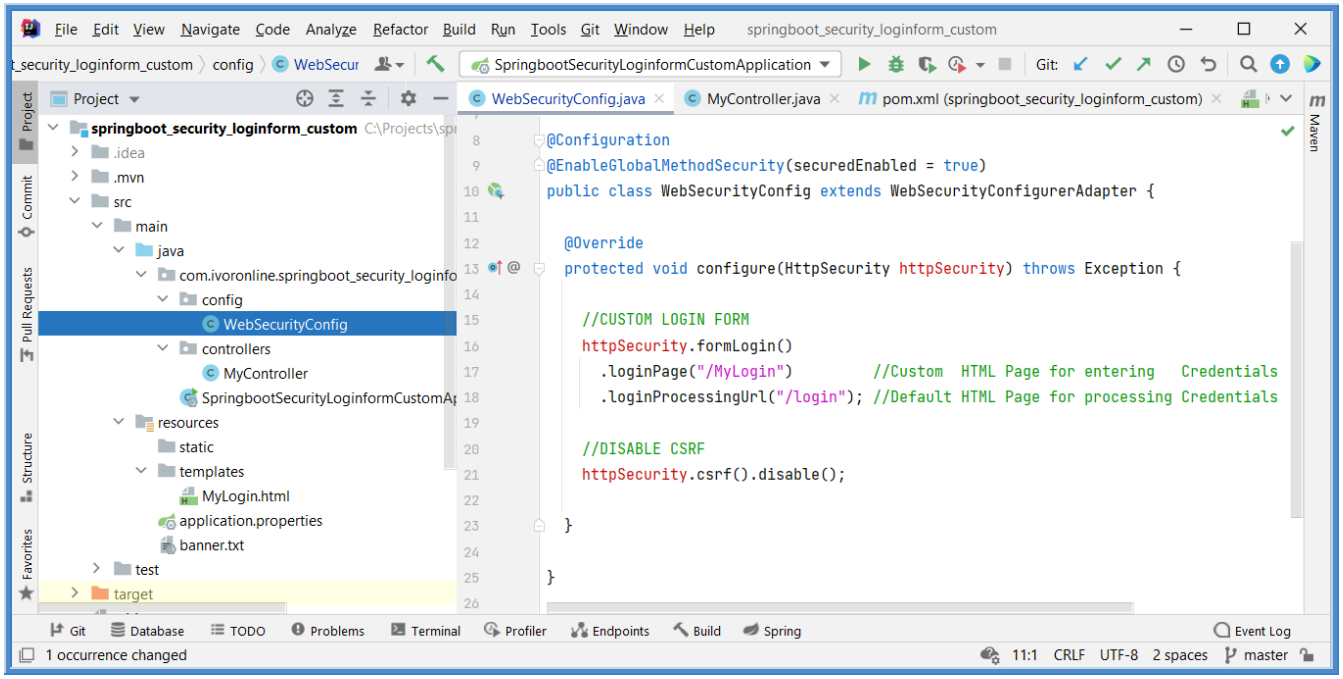
http://localhost:8080/MyLogin



Redirects to http://localhost:8080/Hello



Application Structure



pom.xml

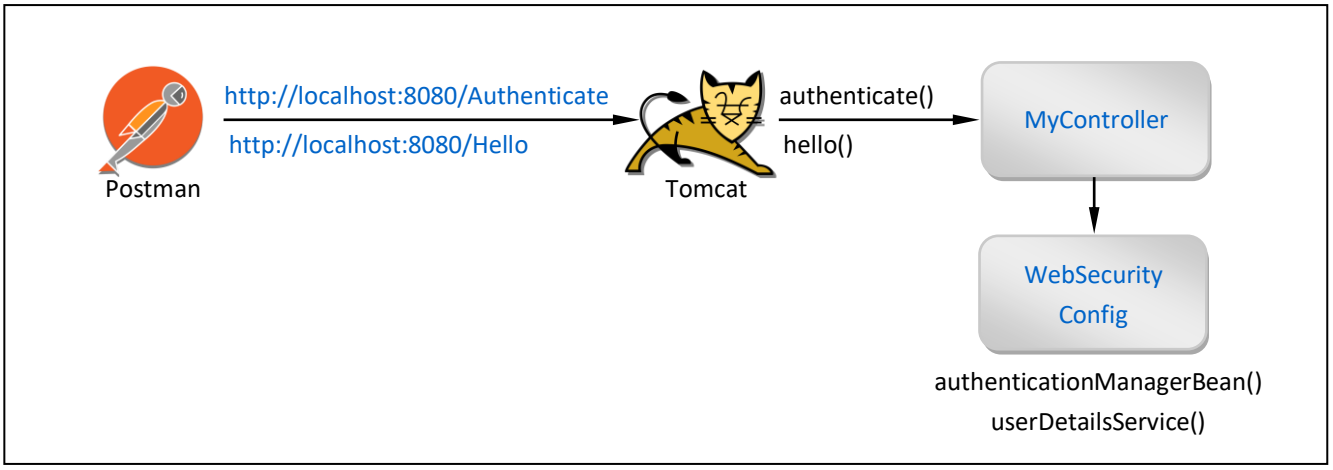
```
<dependencies>

  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</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-thymeleaf</artifactId>
  </dependency>

</dependencies>
```

2.3.3 Manual - Controller - Headers

Info [G]

- This tutorial shows how to **manually Authenticate** User by **manually retrieving Credentials** from **Request Headers**
  http://localhost:8080/Authenticate (GET or POST)
- We will use
  - userDetailsService() (to define Users inside WebSecurityConfig)
  - @Secured("ROLE_ADMIN") (to add Authorities to Endpoint /Hello)
  - **Request Headers** (to read Credentials)
- When Credentials are manually retrieved then you have to (we are doing this inside MyController)
  - manually call AuthenticationManager 's Method authenticate() (to check Password & get Authorities)
  - manually store returned Authentication Object into Context (to allow access to Restricted Resources)
  - after which we can access Restricted Resource http://localhost:8080/Hello

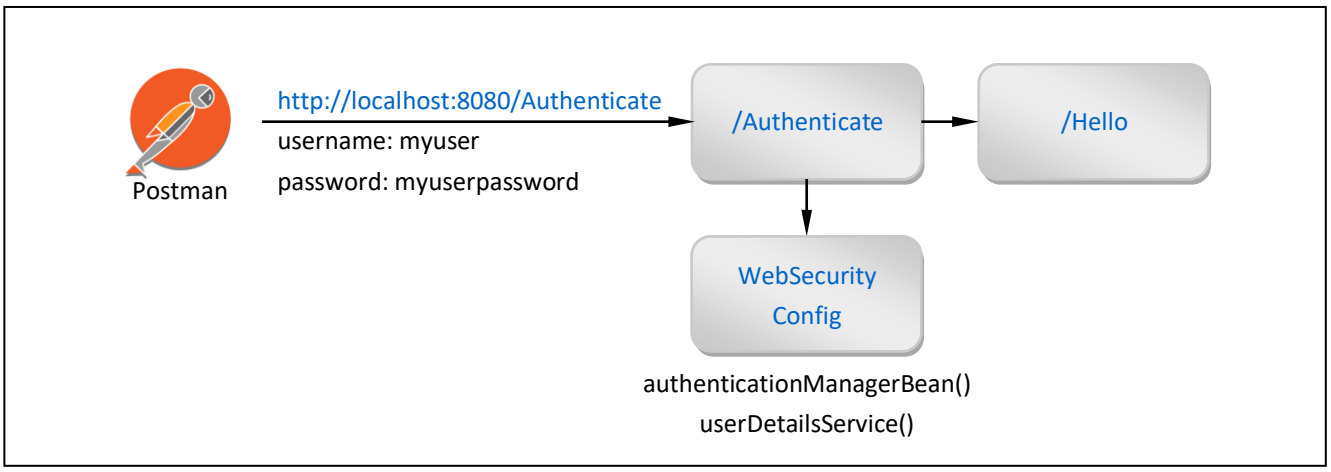
Application Schema *[Results]*



Spring Boot Starters

GROUP	DEPENDENCY	DESCRIPTION
Web	Spring Web	Enables: @RestController, @RequestMapping, Tomcat Server
Security	Spring Security	Enables: @Secured

Authorization Process



Procedure

- Create Project: springboot_security_readcredential_manual_headers (add Spring Boot Starters from the table)
- Create Package: controllers
  - Create Class: MyController.java
- Create Package: config
  - Create Class: WebSecurityConfig.java

MyController.java

```
package com.ivoronline.springboot_security_readcredential_manual_headers.controllers;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.access.annotation.Secured;
import org.springframework.security.authentication.AuthenticationManager;
import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;
import org.springframework.security.core.Authentication;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.web.bind.annotation.RequestHeader;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class MyController {

  @Autowired AuthenticationManager authenticationManager;

  //============================================================================
  // AUTHENTICATE
  //============================================================================
  @RequestMapping("/Authenticate")
  public String authenticate(@RequestHeader String username, @RequestHeader String password) {

    //CREATE AUTHENTICATION OBJECT (with Entered Username & Password)
    Authentication authentication = new UsernamePasswordAuthenticationToken(username, password);

    //GET    AUTHENTICATION OBJECT (with Authorities)
    authentication = authenticationManager.authenticate(authentication);

    //STORE  AUTHENTICATION OBJECT (into Context)
    SecurityContextHolder.getContext().setAuthentication(authentication);

    //RETURN SOMETHING
    return "User Authenticated";

  }

  //============================================================================
  // HELLO
  //============================================================================
  @Secured("ROLE_USER")
  @RequestMapping("/Hello")
  public String hello() {
    return "Hello from Controller";
  }

}
```

WebSecurityConfig.java

```
package com.ivoronline.springboot_security_readcredential_manual_headers.config;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.authentication.AuthenticationManager;
import org.springframework.security.config.annotation.method.configuration.EnableGlobalMethodSecurity;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;
import org.springframework.security.core.userdetails.User;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.crypto.password.NoOpPasswordEncoder;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.security.provisioning.InMemoryUserDetailsManager;

@Configuration
@EnableGlobalMethodSecurity(securedEnabled = true)
public class WebSecurityConfig extends WebSecurityConfigurerAdapter {

  //=====================================================================
  // AUTHENTICATION MANAGER BEAN
  //=====================================================================
  @Bean
  @Override
  public AuthenticationManager authenticationManagerBean() throws Exception {
    return super.authenticationManagerBean();
  }

  //=====================================================================
  // USER DETAILS SERVICE
  //=====================================================================
  @Bean
  @Override
  protected UserDetailsService userDetailsService() {

    //CREATE USERS
    UserDetails myuser  = User.withUsername("myuser" ).password("myuserpassword" ).roles("USER" ).build();
    UserDetails myadmin = User.withUsername("myadmin").password("myadminpassword").roles("ADMIN").build();

    //STORE USERS
    return new InMemoryUserDetailsManager(myuser, myadmin);

  }

  //=====================================================================
  // PASSWORD ENCODER
  //=====================================================================
  @Bean
  PasswordEncoder passwordEncoder() {
    return NoOpPasswordEncoder.getInstance();
  }

  //=====================================================================
  // CONFIGURE
  //=====================================================================
  @Override
  protected void configure(HttpSecurity httpSecurity) throws Exception {
    httpSecurity.authorizeRequests().antMatchers("/Authenticate").permitAll(); //ANONYMOUS ACCESS
    httpSecurity.csrf().disable();                                              //ENABLE POST TO AUTHENTICATE
  }

}
```

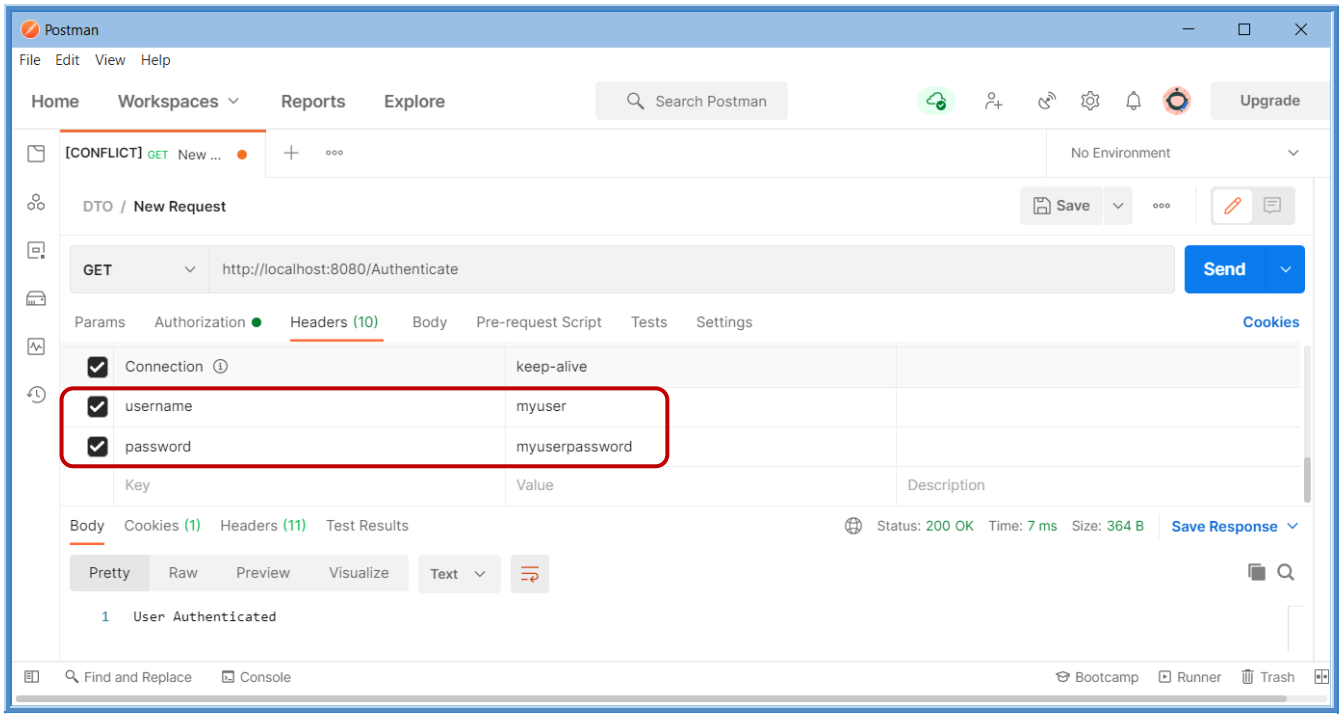
Results

- Start Postman

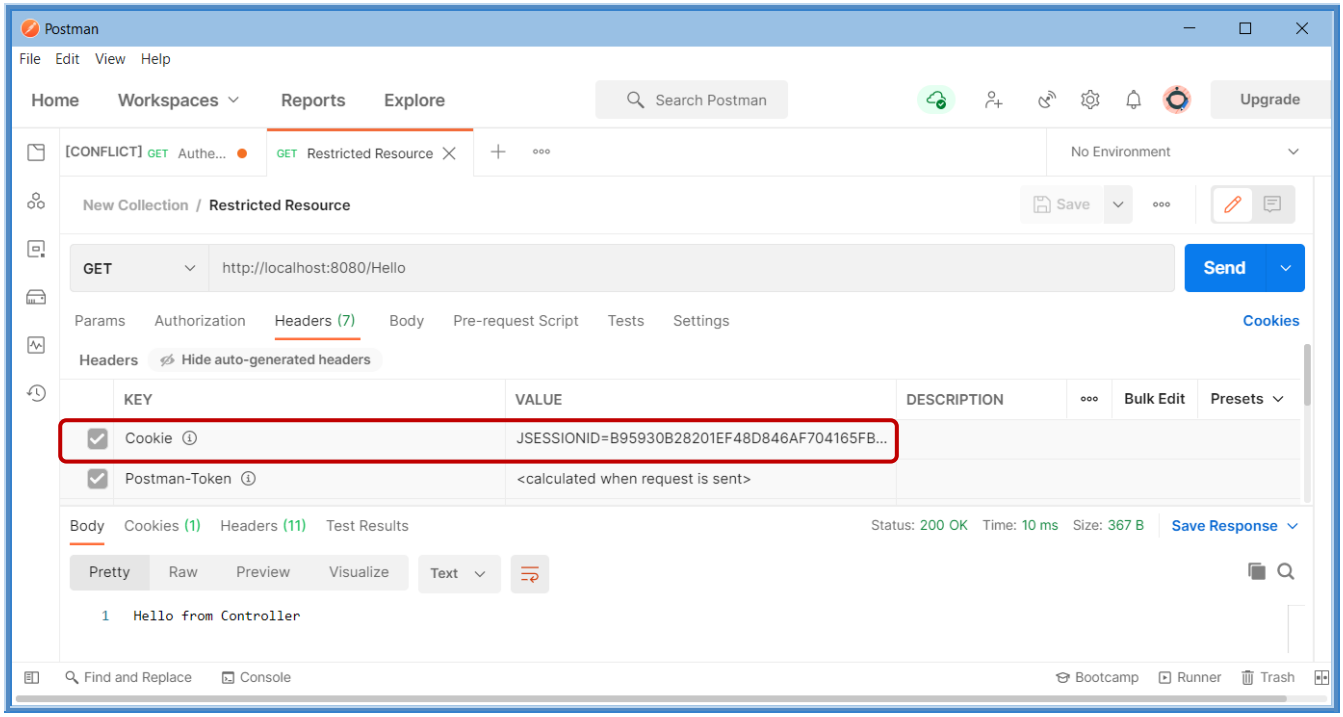
Headers *(add Key-Value)*

```
username: myuser
password: myuserpassword
```

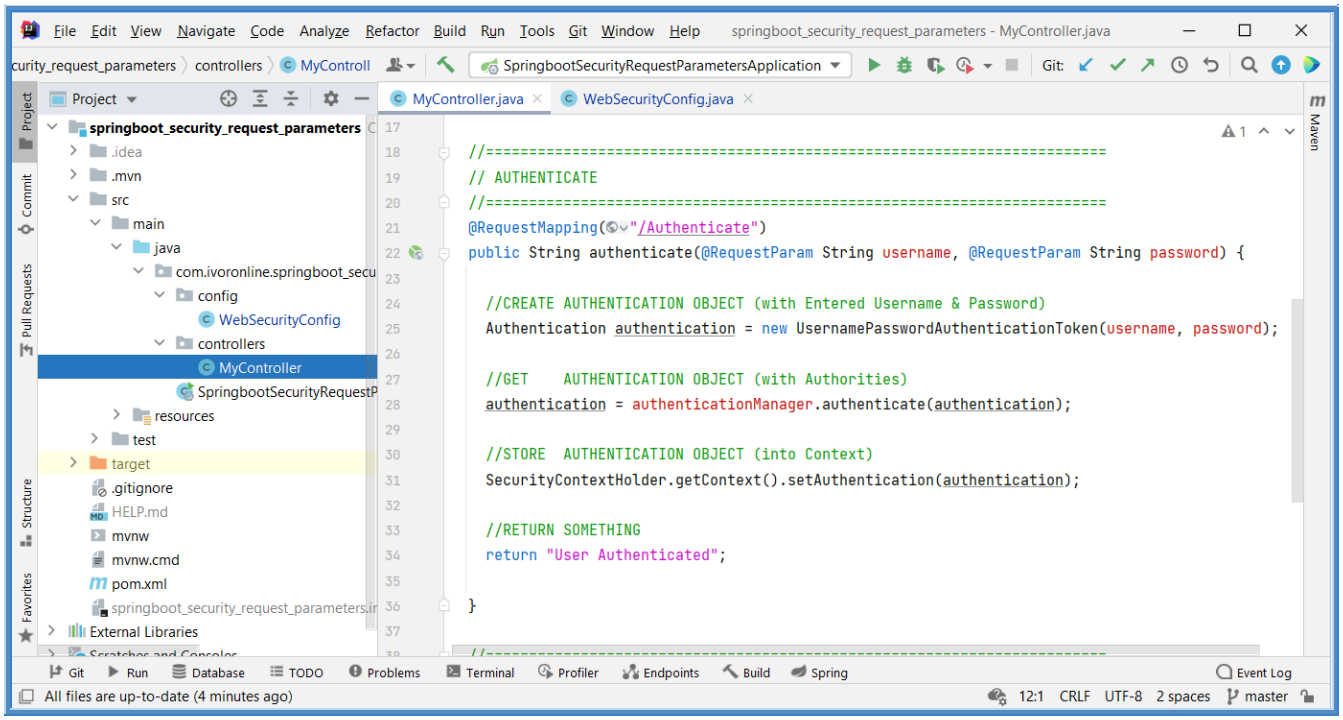
GET or POST http://localhost:8080/Authenticate



GET or POST http://localhost:8080/Hello *(accessible after Authentication by using Cookie JSESSIONID)*



Application Structure



pom.xml

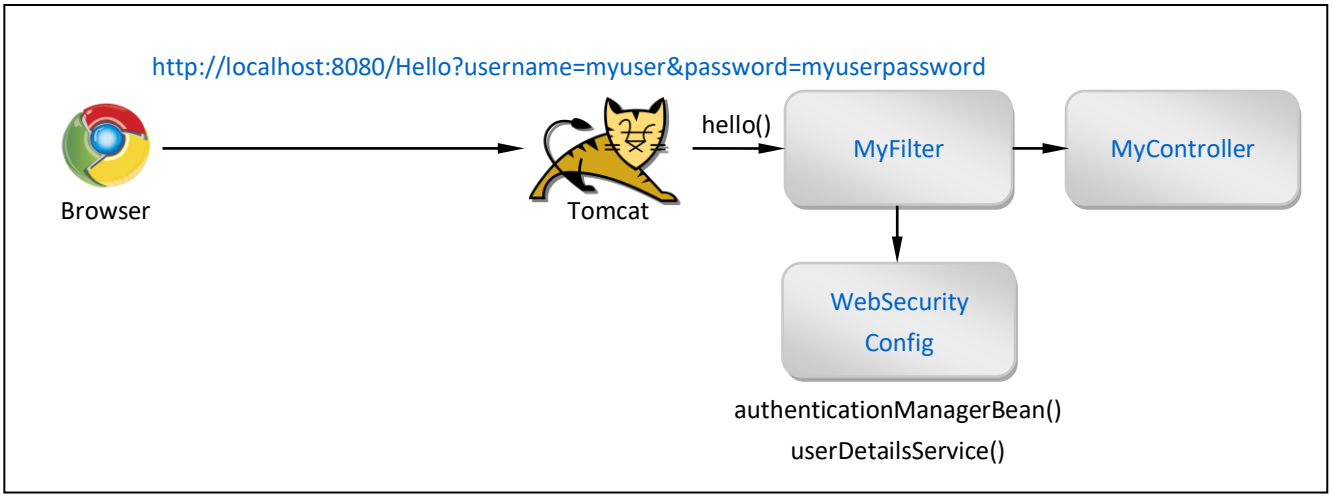
```
<dependencies>

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

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

</dependencies>
```

2.3.4 Manual - Filter - HTTP Request Parameters

Info **[G]**

- This tutorial shows how to **Authenticate User for every HTTP Request** (since we will disable Session) by using **Filter** to
  - read Credentials from HTTP Request Parameters (?username=myuser&password=myuserpassword)
  - call authenticate() with Entered Credentials (to compare them with Stored Credentials)
  - store Authentication Object into Context (for every HTTP Request before it gets forwarded to the Controller)
- Authentication is implemented as shown in tutorial Manual - authenticationManagerBean() - userDetailsService().

Application Schema *[Results]*



Spring Boot Starters

GROUP	DEPENDENCY	DESCRIPTION
Web	Spring Web	Enables: @RestController, @RequestMapping, Tomcat Server
Security	Spring Security	Enables: @Secured

Procedure

- Create Project: springboot_authentication_manual_filter (add Spring Boot Starters from the table)
- Create Package: config (inside main package)
  - Create Class: MyFilter.java (inside config package)
  - Create Class: WebSecurityConfig.java (inside config package)
- Create Package: controllers (inside main package)
  - Create Class: MyController.java (inside controllers package)

MyFilter.java

```
package com.ivoronline.springboot_security_filter_requestparameters.config;

import java.io.IOException;
import javax.servlet.Filter;
import javax.servlet.FilterChain;
import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;
import javax.servlet.http.HttpServletRequest;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.authentication.AuthenticationManager;
import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;
import org.springframework.security.core.Authentication;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.stereotype.Component;

@Component
public class MyFilter implements Filter {

  @Autowired AuthenticationManager authenticationManager;

  //============================================================================
  // AUTHENTICATE
  //============================================================================
  @Override
  public void doFilter(ServletRequest request, ServletResponse response, FilterChain filterchain)
    throws IOException, ServletException {

    //GET CREDENTIALS
    String username = request.getParameter("username");
    String password = request.getParameter("password");

    //CREATE AUTHENTICATION OBJECT (with Entered Username & Password)
    Authentication authentication = new UsernamePasswordAuthenticationToken(username, password);

    //GET    AUTHENTICATION OBJECT (with Authorities)
    authentication = authenticationManager.authenticate(authentication);

    //STORE AUTHENTICATION INTO CONTEXT (SESSION)
    SecurityContextHolder.getContext().setAuthentication(authentication);

    //FORWARD REQUEST
    filterchain.doFilter(request, response);

  }

}
```

WebSecurityConfig.java

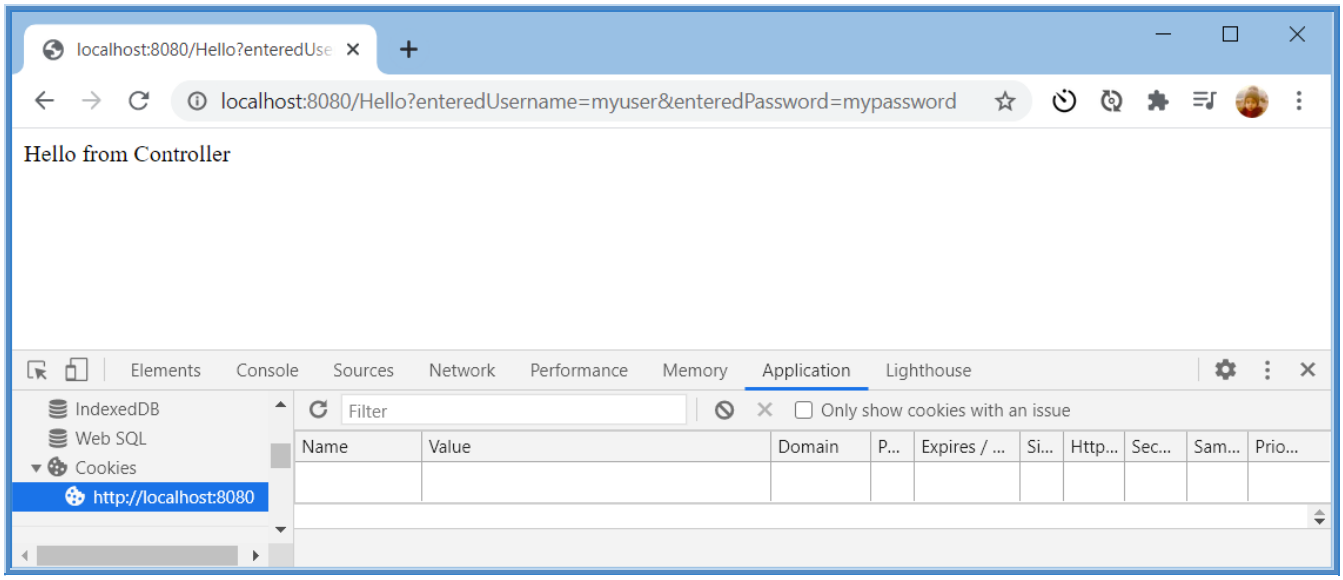
```
package com.ivoronline.springboot_security_filter_requestparameters.config;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.*;
import org.springframework.security.authentication.AuthenticationManager;
import org.springframework.security.config.annotation.method.configuration.EnableGlobalMethodSecurity;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;
import org.springframework.security.config.http.SessionCreationPolicy;
import org.springframework.security.core.userdetails.*;
import org.springframework.security.crypto.password.*;
import org.springframework.security.provisioning.InMemoryUserDetailsManager;
import org.springframework.security.web.authentication.UsernamePasswordAuthenticationFilter;

@Configuration
@EnableGlobalMethodSecurity(securedEnabled = true)
public class WebSecurityConfig extends WebSecurityConfigurerAdapter {

  @Autowired MyFilter myFilter;

  //=====================================================================
  // AUTHENTICATION MANAGER BEAN
  //=====================================================================
  @Bean
  @Override
  public AuthenticationManager authenticationManagerBean() throws Exception {
    return super.authenticationManagerBean();
  }

  //=====================================================================
  // USER DETAILS SERVICE
  //=====================================================================
  @Bean
  @Override
  protected UserDetailsService userDetailsService() {

    //CREATE USERS
    UserDetails myuser  = User.withUsername("myuser" ).password("myuserpassword" ).roles("USER" ).build();
    UserDetails myadmin = User.withUsername("myadmin").password("myadminpassword").roles("ADMIN").build();

    //STORE USER
    return new InMemoryUserDetailsManager(myuser, myadmin);

  }

  //=====================================================================
  // PASSWORD ENCODER
  //=====================================================================
  @Bean
  PasswordEncoder passwordEncoder() {
    return NoOpPasswordEncoder.getInstance();
  }

  //=====================================================================
  // CONFIGURE
  //=====================================================================
  @Override
  protected void configure(HttpSecurity httpSecurity) throws Exception {
    httpSecurity.sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS); //No Session
    httpSecurity.csrf().disable();                                                           //Enable POST
  }

}
```

MyController.java

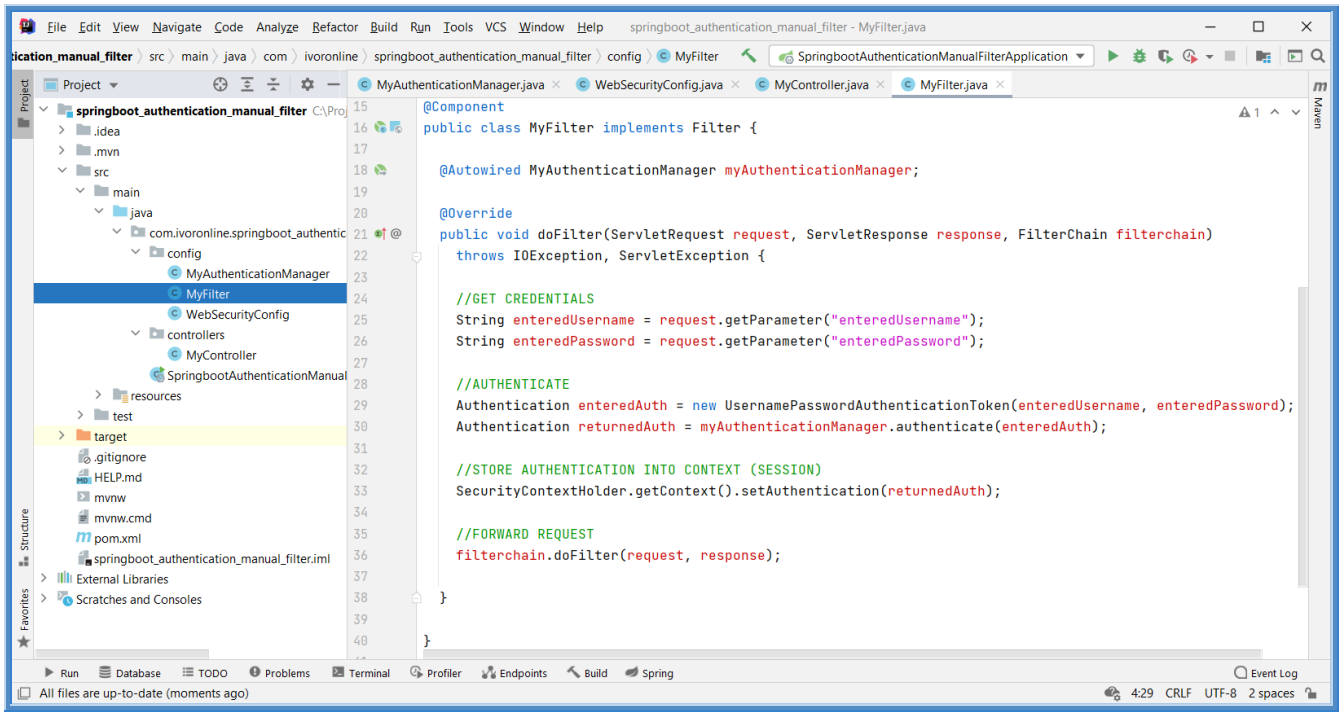
```
package com.ivoronline.springboot_security_filter_requestparameters.controllers;

import org.springframework.security.access.annotation.Secured;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class MyController {

  //=========================================================================
  // HELLO
  //=========================================================================
  @Secured("ROLE_USER")
  @RequestMapping("/Hello")
  public String hello() {
    return "Hello from Controller";
  }

}
```

Results

http://localhost:8080/Hello?username=myuser&password=myuserpassword *(no Session Cookie)*



Application Structure



pom.xml

```
<dependencies>

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

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

</dependencies>
```

2.4 Authentication

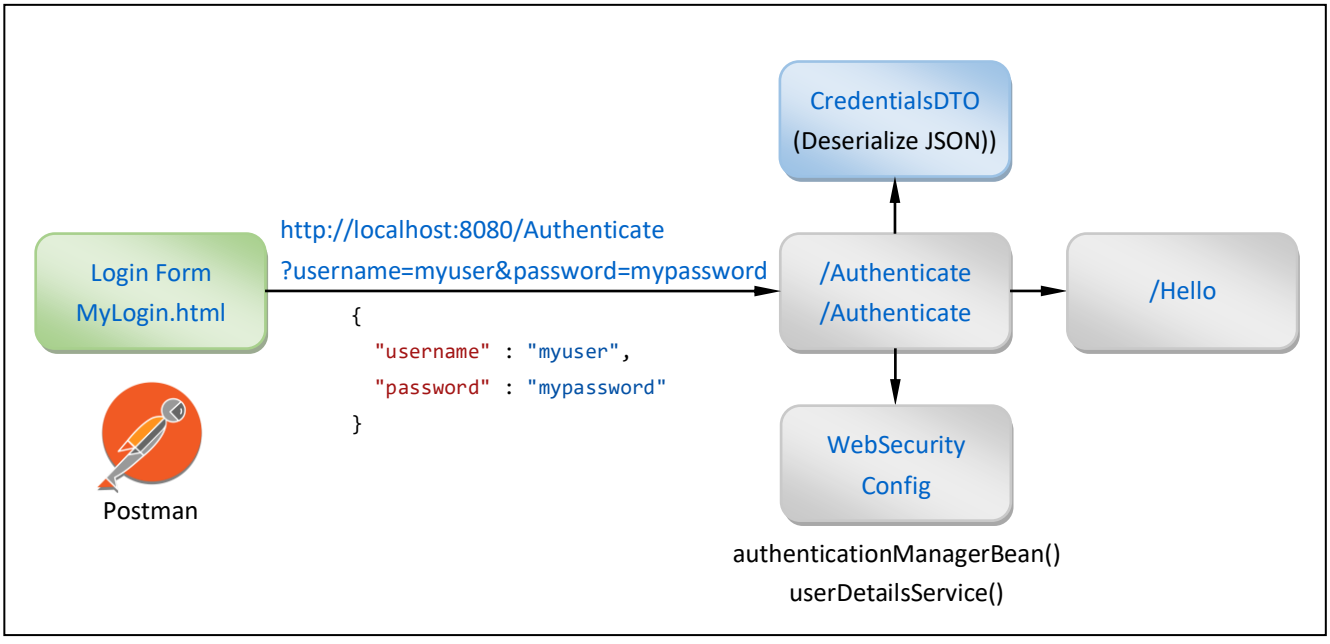
Info

- Following tutorials show different combinations of Authenticating and storing Users.
- In tutorials named with
  - **Automatic** Spring Framework automatically reads User Credentials and calls AuthenticationManger
  - **Manually** we manually read User Credentials and then we manually call AuthenticationManger
- For each of these two types of tutorials we show different ways of defining Users by using either
  - File **application.properties**
  - Method **configure(Authentication...)** in Class WebSecurityConfig
  - Method **userDetailsService()** in Class WebSecurityConfig
  - Class **MyUserDetailsService**
- For manual Authentication use either
  - Method **authenticationManagerBean()** in Class WebSecurityConfig
  - Class **MyAuthenticationManager**

Manually Read Credentials From HTTP Request

- These tutorials show three different ways of providing Credentials
  - HTTP Request - Parameters (Browser or Postman generates HTTP Request POST or GET)
  - HTTP Request - Parameters - Login Form (Login Form generates HTTP Request POST or GET)
  - HTTP Request - JSON Body (Postman generates HTTP Request POST with JSON Body)
- In all cases HTTP Request Parameters/JSON are manually read in the Controller and WebSecurityConfig is used for Authentication through authenticationManagerBean() and userDetailsService().

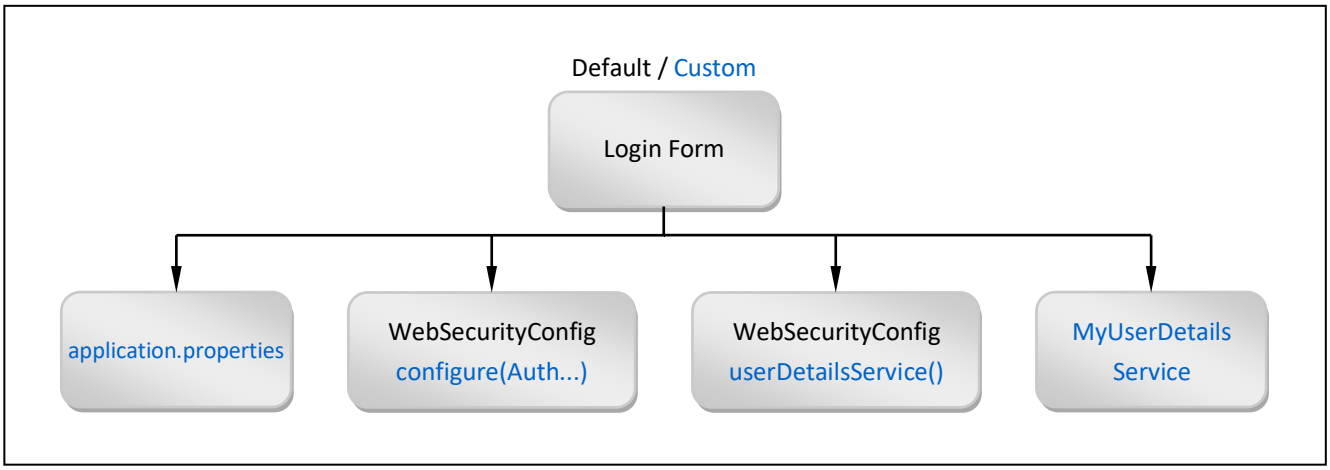
Schema *(Use Postman to create HTTP POST Request with JSON Body)*



Authentication - Automatic

- These tutorials show different ways of providing Users
  - when using Default or Custom Login Form
  - from which Spring Framework automatically reads User Credentials and calls AuthenticationManger
- For Custom Login Form only example using application.properties is demonstrated but Users can also be provided in all the other ways demonstrated for Default Login Form.

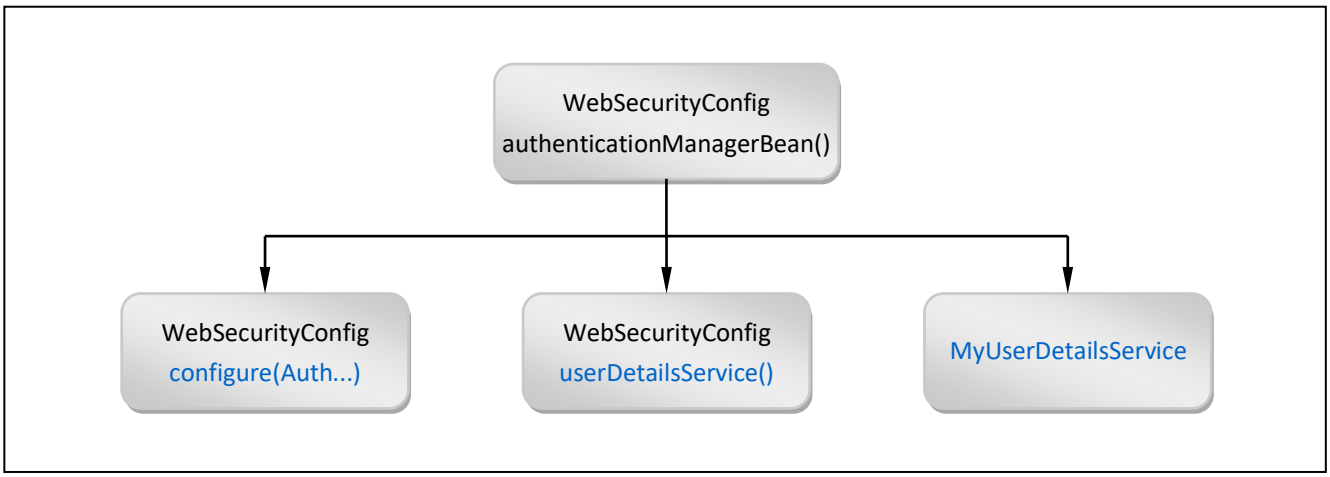
Automatic Authentication



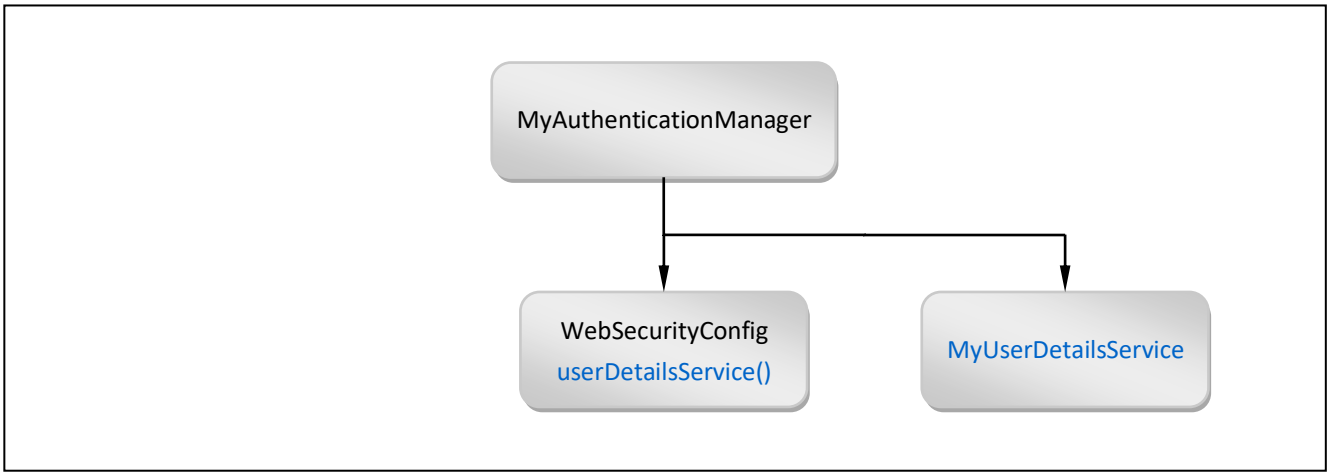
Authentication - Manual

- These tutorials show to manually read User Credentials from HTTP Request Parameters and then
  - different ways of manually calling AuthenticationManger
  - different ways of providing Users

authenticationManagerBean()



MyAuthenticationManager

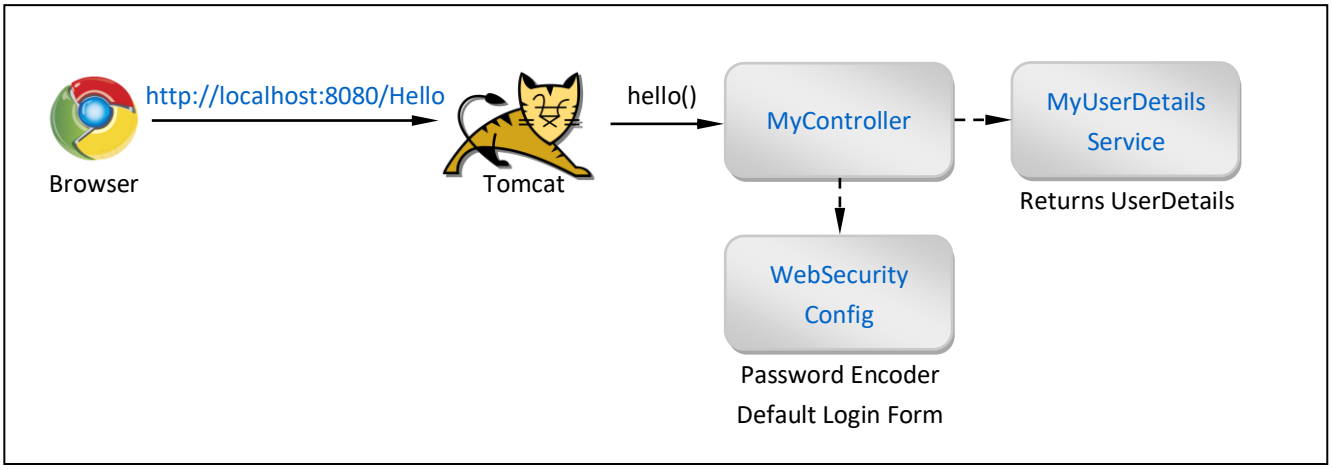


2.4.1 Automatic - Login Form - Default - MyUserDetailsService

Info [G]

- This tutorial shows how to use
  - **MyUserDetailsService** (to return hard coded UserDetails for Entered Username)
  - @Secured("ROLE_ADMIN") (to add Authorities to Endpoint /Hello)
  - Default Login Form (to read Credentials)

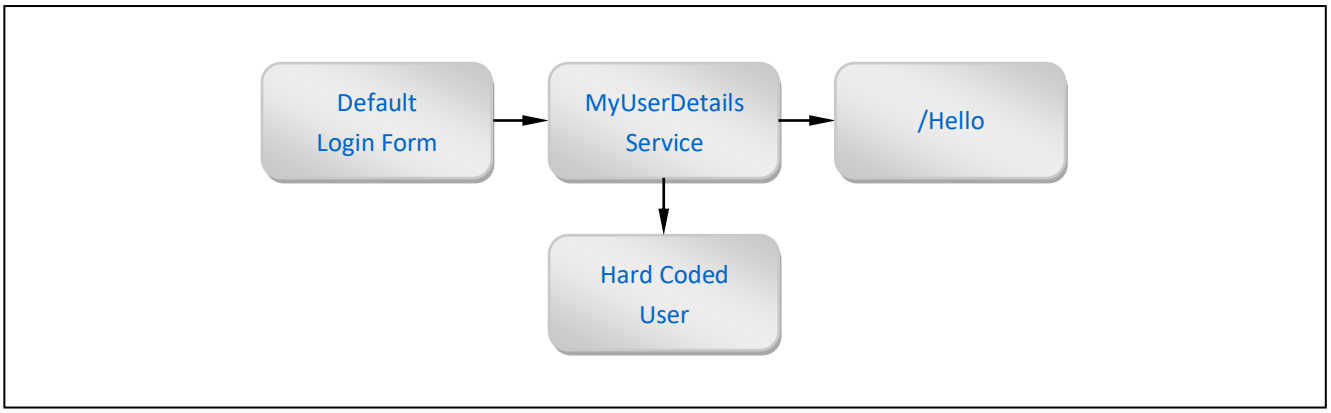
Application Schema *[Results]*



Spring Boot Starters

GROUP	DEPENDENCY	DESCRIPTION
Web	Spring Web	Enables: @RestController, @RequestMapping, Tomcat Server
Security	Spring Security	Enables: @Secured

Authorization Process



Procedure

- Create Project: springboot_manuallycreateuserobject (add Spring Boot Starters from the table)
- Create Package: controllers (inside main package)
  - Create Class: MyController.java (inside package controllers)
- Create Package: services (inside main package)
  - Create Class: MyUserDetailsService.java (inside package services)
- Create Package: config (inside main package)
  - Create Class: WebSecurityConfig.java (inside package config)

MyUserDetailsService.java

```
package com.ivoronline.springboot_manuallycreateuserobject.services;

import org.springframework.security.core.GrantedAuthority;
import org.springframework.security.core.authority.SimpleGrantedAuthority;
import org.springframework.security.core.userdetails.User;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.core.userdetails.UsernameNotFoundException;
import org.springframework.stereotype.Service;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;

@Service
public class MyUserDetailsService implements UserDetailsService {

  @Override
  public UserDetails loadUserByUsername(String enteredUsername) throws UsernameNotFoundException {

    //-----------------------------------------------------------------------------------------------
    //MOCK DB: <username, {"password", "ROLE"}>
    HashMap<String, String[]> accounts = new HashMap<>();
                              accounts.put("myuser" , new String[]{"myuserpassword" , "ROLE_USER" });
                              accounts.put("myadmin", new String[]{"myadminpassword", "ROLE_ADMIN"});
    //-----------------------------------------------------------------------------------------------

    //GET USER/ACCOUNT (From DB)
    String[] account = accounts.get(enteredUsername);

    //CHECK IF USER EXISTS
    if (account == null) { throw new UsernameNotFoundException(enteredUsername); }  //Bad credentials

    //GET PASSWORD
    String storedPassword = account[0];

    //GET AUTHORITIES
    List<GrantedAuthority> authorities = new ArrayList<GrantedAuthority>();
                           authorities.add(new SimpleGrantedAuthority(account[1]));

    //CREATE USER DETAILS OBJECT
    UserDetails userDetails = new User(enteredUsername, storedPassword, authorities);

    //RETURN USER
    return userDetails;

  }

}
```

WebSecurityConfig.java

```
package com.ivoronline.springboot_manuallycreateuserobject.config;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.config.annotation.method.configuration.EnableGlobalMethodSecurity;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;
import org.springframework.security.crypto.password.NoOpPasswordEncoder;
import org.springframework.security.crypto.password.PasswordEncoder;

@Configuration
@EnableGlobalMethodSecurity(securedEnabled = true)
public class WebSecurityConfig extends WebSecurityConfigurerAdapter {

  //=========================================================================
  // PASSWORD ENCODER
  //=========================================================================
  @Bean
  PasswordEncoder passwordEncoder() {
    return NoOpPasswordEncoder.getInstance();
  }

  //=========================================================================
  // CONFIGURE
  //=========================================================================
  @Override
  protected void configure(HttpSecurity httpSecurity) throws Exception {
    httpSecurity.authorizeRequests().antMatchers("/Authenticate").permitAll(); //ANONYMOUS ACCESS
    httpSecurity.formLogin();                                                   //DEFAULT LOGIN FORM
  }

}
```

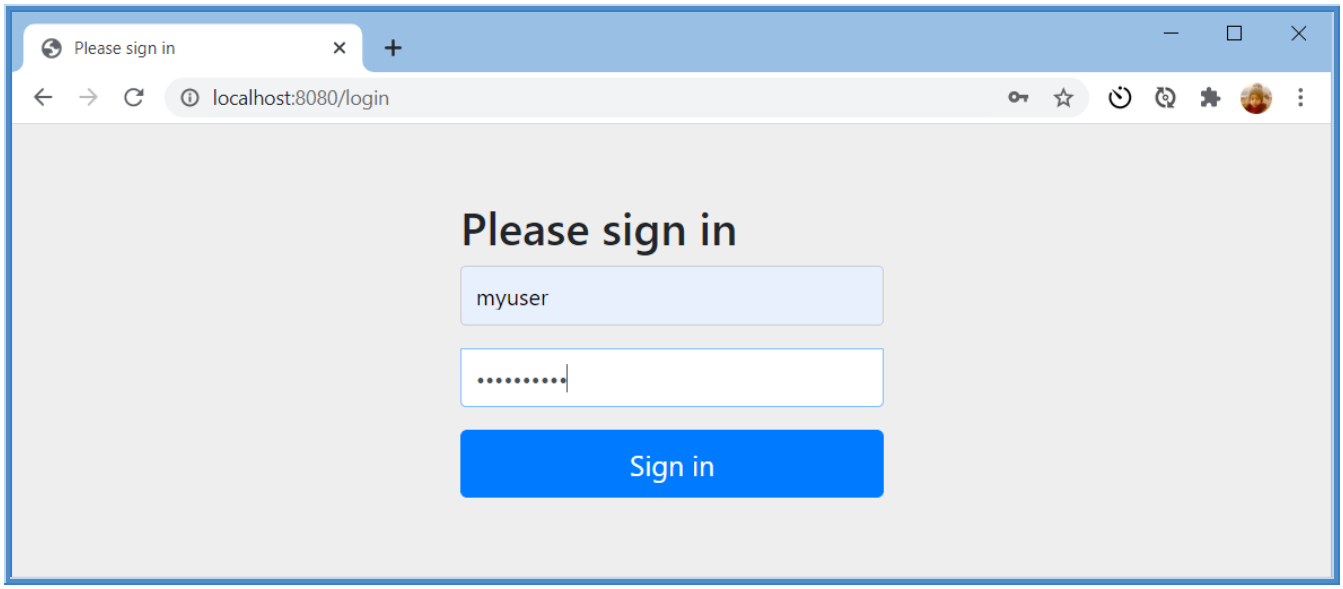
MyController.java

```
package com.ivoronline.springboot_manuallycreateuserobject.controllers;

import org.springframework.security.access.annotation.Secured;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class MyController {

  //=============================================================================
  // HELLO (Secured Resource)
  //=============================================================================
  @Secured("ROLE_USER")
  @RequestMapping("/Hello")
  public String hello() {
    return "Hello from Controller";
  }

}
```

Results

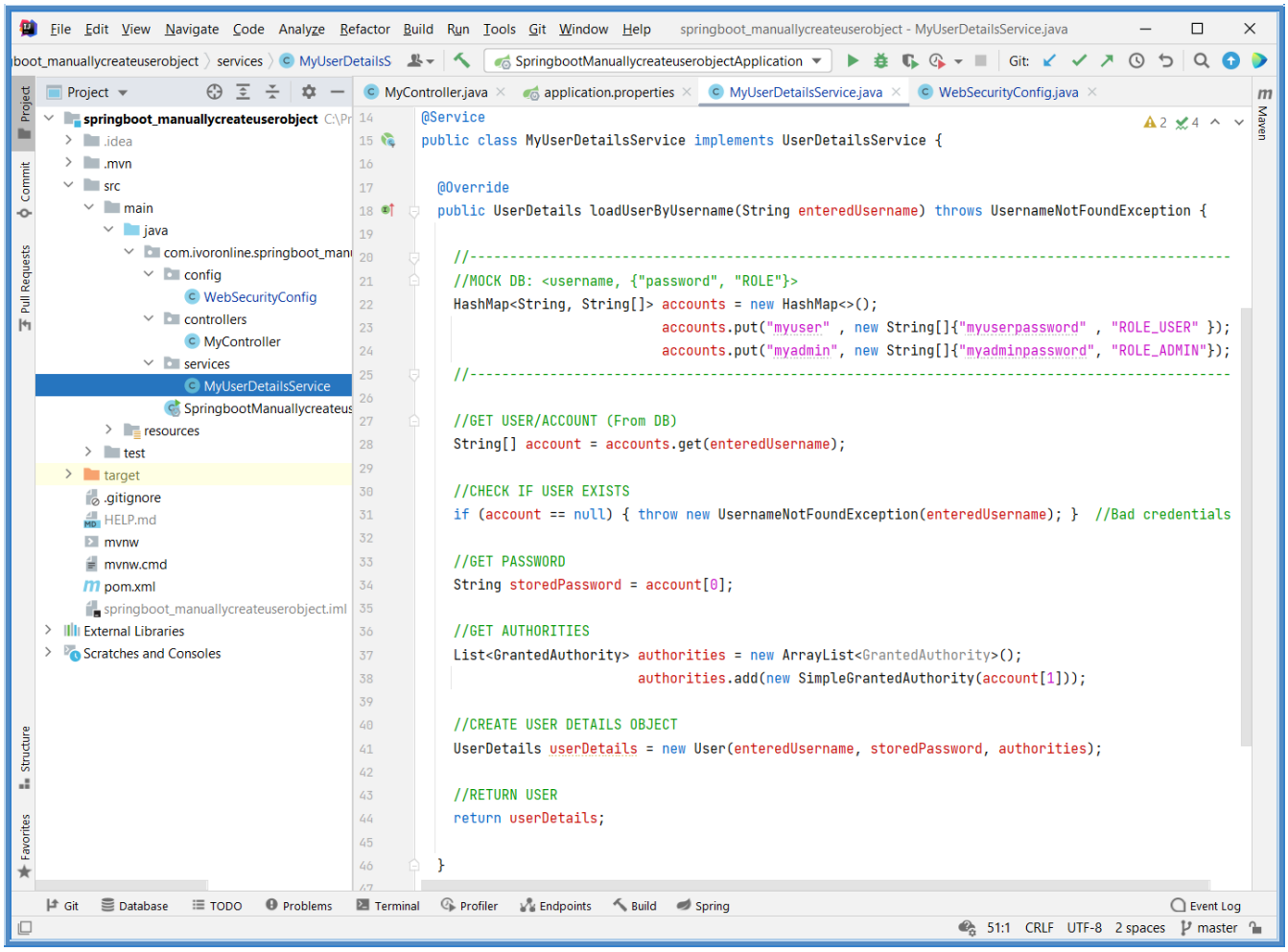
- http://localhost:8080/Hello
- You get redirected to http://localhost:8080/login
  - Username: **myuser**
  - Password: **myuserpassword**
  - Sign in
- You get redirected back to http://localhost:8080/Hello

http://localhost:8080/login - myuser - myuserpassword



Redirects to http://localhost:8080/Hello

Application Structure



pom.xml

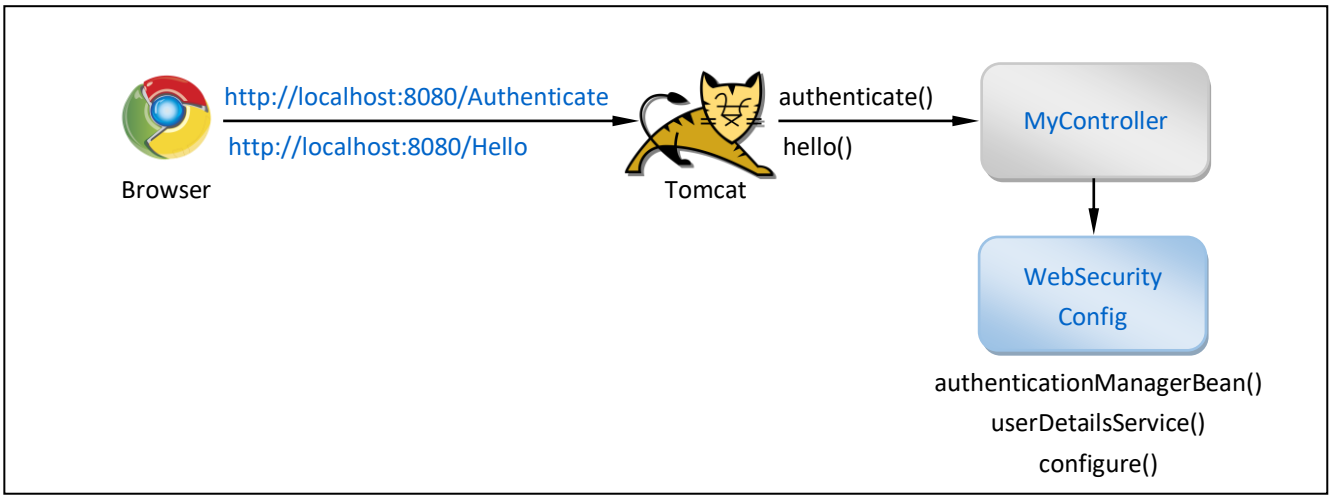
```
<dependencies>

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

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

</dependencies>
```

2.4.2 Manual - authenticationManagerBean() - userDetailsService()

Info [G]

- This tutorial shows how to **manually Authenticate** User by **manually retrieving Credentials** from **Request Parameters**
  http://localhost:8080/Authenticate?username=myuser&password=mypassword
- When you manually retrieve Credentials you have to manually call authenticate() and in this tutorials
  - we manually call authenticate() from authenticationManagerBean() (to check Password & get Authorities)
  - which calls loadUserByUsername() from userDetailsService() (to get Stored Password & Authorities)
  - we manually store returned Authentication Object into Context (to allow access to Restricted Resource)
  - after which we can access Restricted Resource http://localhost:8080/Hello

Application Schema *[Results]*



Spring Boot Starters

GROUP	DEPENDENCY	DESCRIPTION
Web	Spring Web	Enables: @RestController, @RequestMapping, Tomcat Server
Security	Spring Security	Enables: @Secured

Procedure

- Create Project: springboot_security_manual1 (add Spring Boot Starters from the table)
- Create Package: config (inside main package)
  - Create Class: WebSecurityConfig.java (inside config package)
- Create Package: controllers (inside main package)
  - Create Class: MyController.java (inside controllers package)

WebSecurityConfig.java

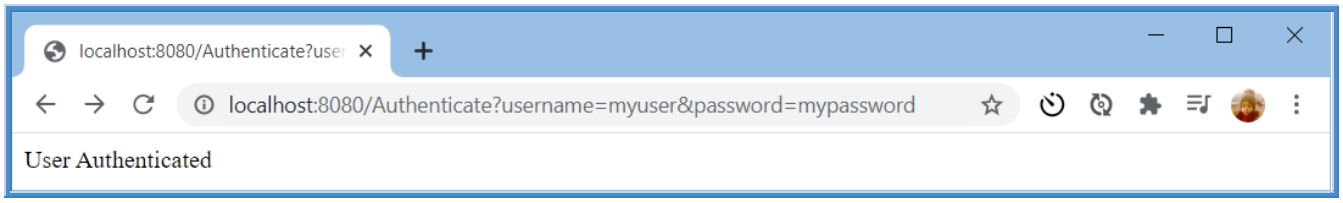
```
package com.ivoronline.springboot_security_manual1.config;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.authentication.AuthenticationManager;
import org.springframework.security.config.annotation.method.configuration.EnableGlobalMethodSecurity;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;
import org.springframework.security.core.userdetails.*;
import org.springframework.security.provisioning.InMemoryUserDetailsManager;

@Configuration
@EnableGlobalMethodSecurity(securedEnabled = true)
public class WebSecurityConfig extends WebSecurityConfigurerAdapter {

  //=================================================================
  // AUTHENTICATION MANAGER BEAN
  //=================================================================
  @Bean
  @Override
  public AuthenticationManager authenticationManagerBean() throws Exception {
    return super.authenticationManagerBean();
  }

  //=================================================================
  // USER DETAILS SERVICE
  //=================================================================
  @Bean
  @Override
  protected UserDetailsService userDetailsService() {

    //CREATE USERS
    UserDetails myuser  = User.withUsername("myuser" ).password("myuserpassword" ).roles("USER" ).build();
    UserDetails myadmin = User.withUsername("myadmin").password("myadminpassword").roles("ADMIN").build();

    //STORE USER
    return new InMemoryUserDetailsManager(myuser, myadmin);

  }

  //=================================================================================
  // PASSWORD ENCODER
  //=================================================================================
  @Bean
  PasswordEncoder passwordEncoder() {
    return NoOpPasswordEncoder.getInstance();
  }

  //=================================================================
  // CONFIGURE
  //=================================================================
  @Override
  protected void configure(HttpSecurity httpSecurity) throws Exception {
    httpSecurity.authorizeRequests().antMatchers("/Authenticate").permitAll(); //ANONYMOUS ACCESS
    httpSecurity.csrf().disable();                                              //ENABLE POST TO AUTHENTICATE
  }

}
```

MyController.java

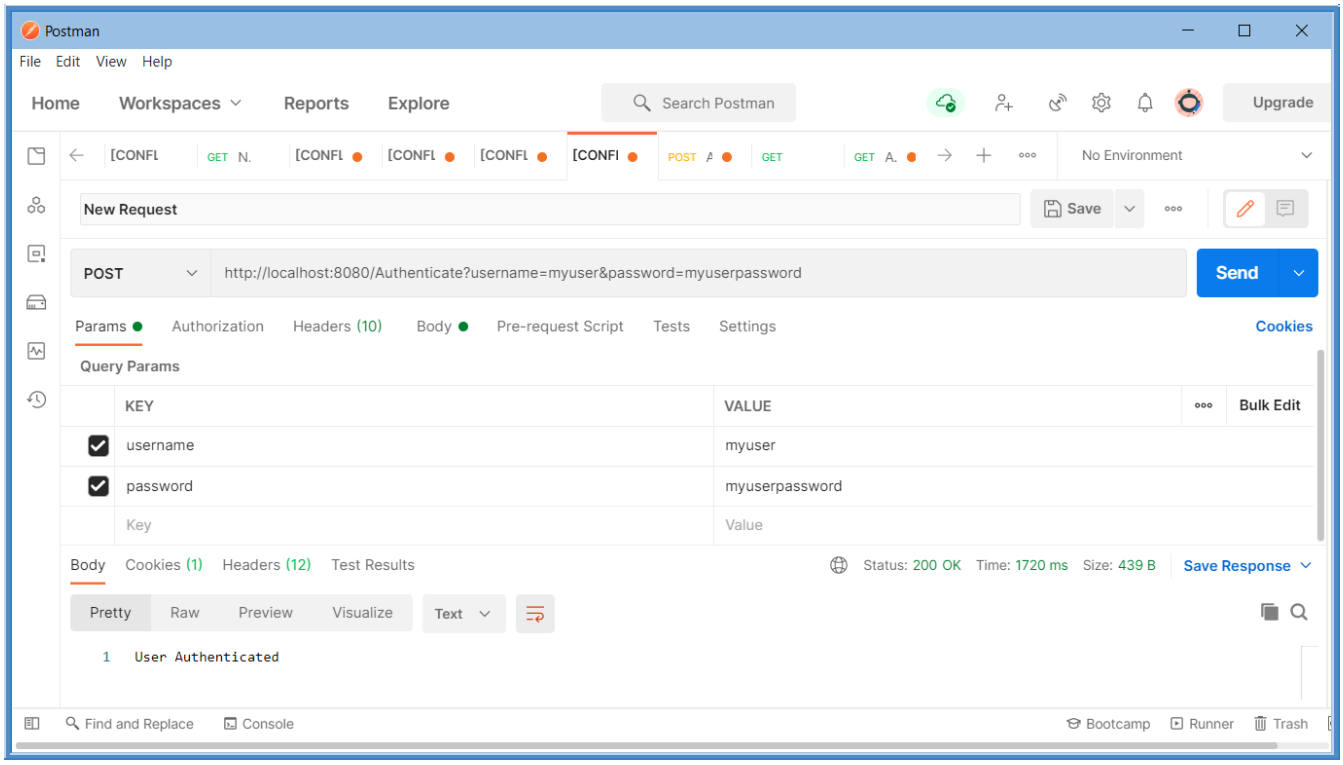
```java
package com.ivoronline.springboot_security_manual1.controllers;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.access.annotation.Secured;
import org.springframework.security.authentication.AuthenticationManager;
import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;
import org.springframework.security.core.Authentication;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class MyController {

  @Autowired AuthenticationManager authenticationManager;

  //===============================================================================
  // AUTHENTICATE
  //===============================================================================
  @RequestMapping("/Authenticate")
  public String authenticate(@RequestParam String username, @RequestParam String password) {

    //CREATE AUTHENTICATION OBJECT (with Entered Username & Password)
    Authentication authentication = new UsernamePasswordAuthenticationToken(username, password);

    //GET    AUTHENTICATION OBJECT (with Authorities)
    authentication = authenticationManager.authenticate(authentication);

    //STORE  AUTHENTICATION OBJECT (into Context)
    SecurityContextHolder.getContext().setAuthentication(authentication);

    //RETURN SOMETHING
    return "User Authenticated";

  }

  //===============================================================================
  // HELLO
  //===============================================================================
  @Secured("ROLE_USER")
  @RequestMapping("/Hello")
  public String hello() {
    return "Hello from Controller";
  }

}
```

Results

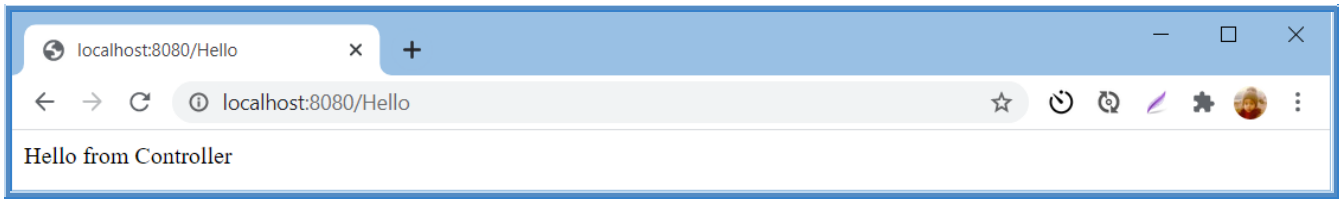
http://localhost:8080/Authenticate?username=myuser&password=myuserpassword *(GET)*



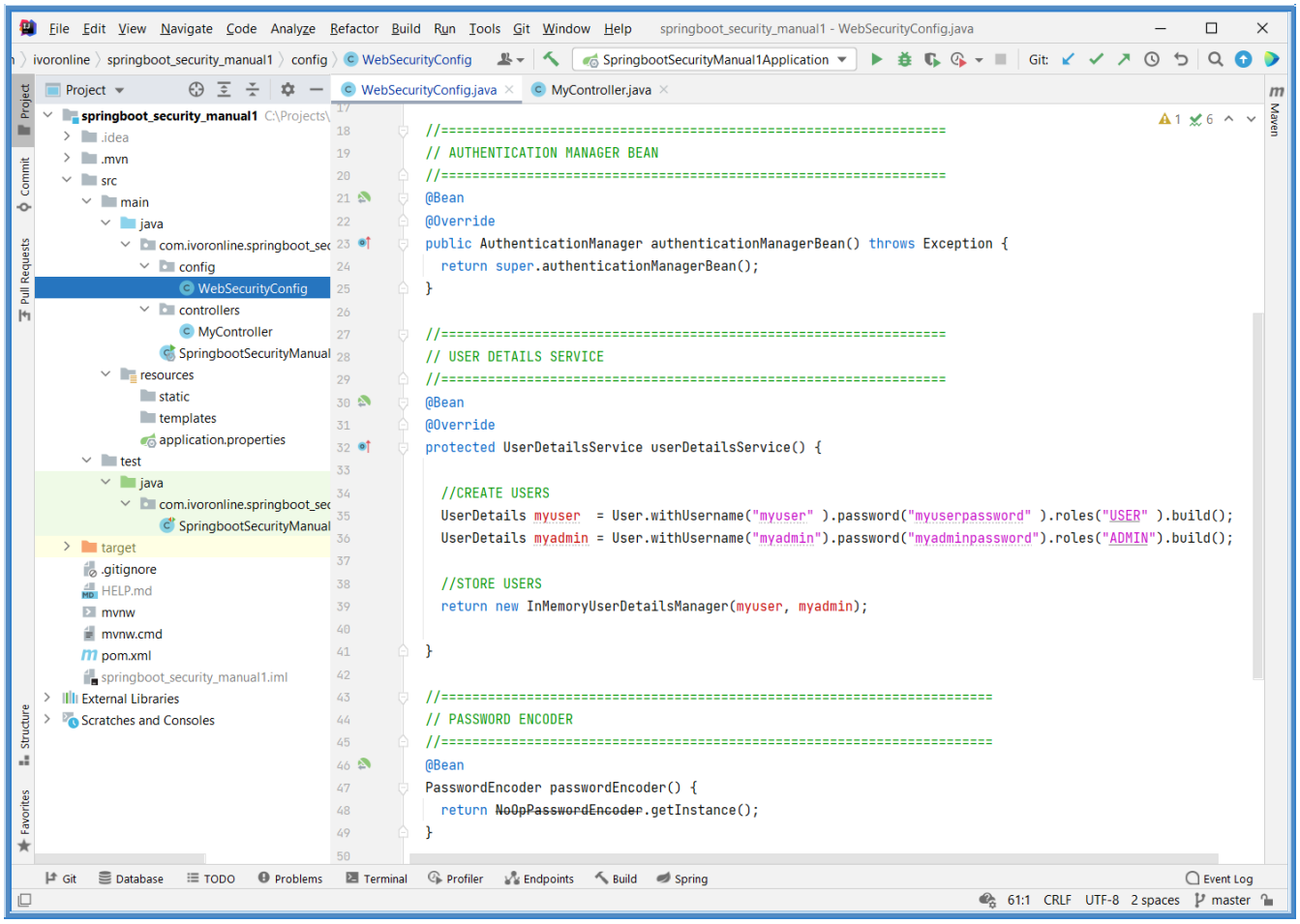
http://localhost:8080/Authenticate?username=myuser&password=myuserpassword *(POST)*



http://localhost:8080/Hello *(accessible only after authentication)*



Application Structure



pom.xml

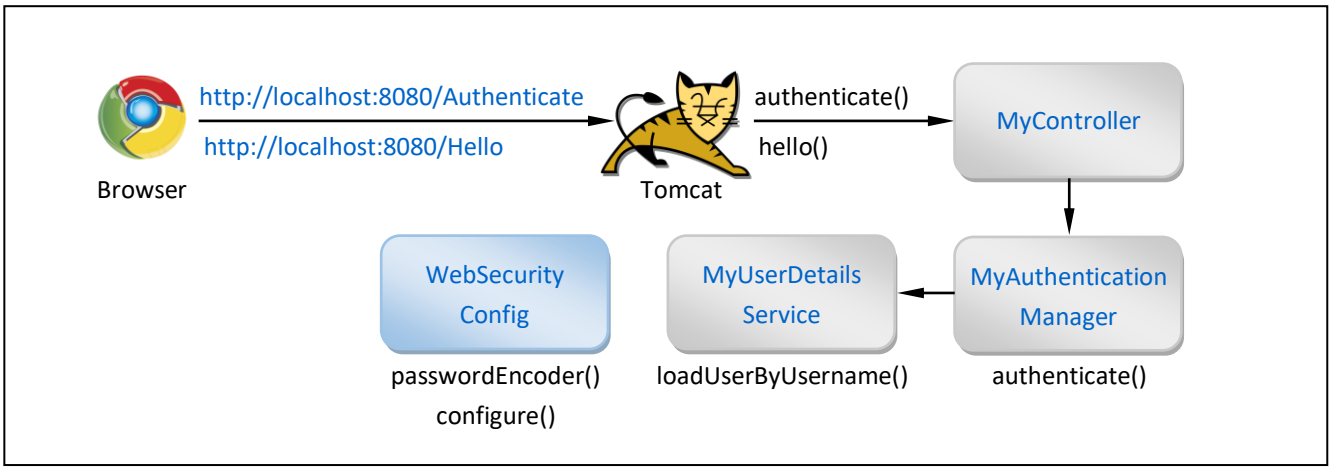
```
<dependencies>

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

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

</dependencies>
```

2.4.3 Manual - MyAuthenticationManager - MyUserDetailsService

Info **[G]**

- This tutorial shows how to **manually Authenticate** User by **manually retrieving Credentials** from **Request Parameters**
  http://localhost:8080/Authenticate?username=myuser&password=mypassword
- When you manually retrieve Credentials you have to manually call `authenticate()` and in this tutorials
  - we manually call `authenticate()` from `MyAuthenticationManager` (to check Password & get Authorities)
  - which calls `loadUserByUsername()` from `MyUserDetailsService` (to get Stored Password & Authorities)
  - we manually store returned `Authentication` Object into Context (to allow access to Restricted Resource)
  - after which we can access Restricted Resource http://localhost:8080/Hello

Application Schema *[Results]*



Spring Boot Starters

GROUP	DEPENDENCY	DESCRIPTION
Web	Spring Web	Enables: @RestController, @RequestMapping, Tomcat Server
Security	Spring Security	Enables: @Secured
Developer Tools	Lombok	Enables: @Slf4j

Procedure

- Create Project: springboot_security_manual3 (add Spring Boot Starters from the table)
- Create Package: config (inside main package)
  - Create Class: WebSecurityConfig.java (inside config package)
- Create Package: services (inside main package)
  - Create Class: MyUserDetailsService.java (inside services package)
  - Create Class: MyAuthenticationManager.java (inside services package)
- Create Package: controllers (inside main package)
  - Create Class: MyController.java (inside controllers package)

WebSecurityConfig.java

```
package com.ivoronline.springboot_security_manual3.config;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.config.annotation.method.configuration.EnableGlobalMethodSecurity;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;
import org.springframework.security.crypto.password.NoOpPasswordEncoder;
import org.springframework.security.crypto.password.PasswordEncoder;

@Configuration
@EnableGlobalMethodSecurity(securedEnabled = true)
public class WebSecurityConfig extends WebSecurityConfigurerAdapter {

  //=========================================================================
  // PASSWORD ENCODER
  //=========================================================================
  @Bean
  PasswordEncoder passwordEncoder() {
    return NoOpPasswordEncoder.getInstance();
  }

  //=========================================================================
  // CONFIGURE
  //=========================================================================
  @Override
  protected void configure(HttpSecurity httpSecurity) throws Exception {
    httpSecurity.authorizeRequests().antMatchers("/Authenticate").permitAll(); //ANONYMOUS ACCESS
    httpSecurity.csrf().disable();                                              //ENABLE POST TO AUTHENTICATE
  }

}
```

MyAuthenticationManager.java

```java
package com.ivoronline.springboot_security_manual3.services;

import lombok.extern.slf4j.Slf4j;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.authentication.AuthenticationManager;
import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;
import org.springframework.security.core.Authentication;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.stereotype.Component;

@Slf4j
@Component
public class MyAuthenticationManager implements AuthenticationManager {

  @Autowired UserDetailsService userDetailsService;

  @Override
  public Authentication authenticate(Authentication enteredAuthentication) {

    //GET ENTERED CREDENTIALS
    String enteredUsername = (String) enteredAuthentication.getPrincipal();   //USERNAME
    String enteredPassword = (String) enteredAuthentication.getCredentials(); //PASSWORD

    //GET USER DETAILS
    UserDetails userDetails = userDetailsService.loadUserByUsername(enteredUsername);

    //CHECK USER DETAILS
    if ( userDetails == null                              ) { log.error("Username not found"); return null; }
    if (!enteredPassword.equals(userDetails.getPassword())) { log.error("Incorrect Password"); return null; }

    //CREATE VALIDATED AUTHENTICATION OBJECT
    Authentication authentication = new UsernamePasswordAuthenticationToken(
      enteredUsername,
      enteredPassword ,
      userDetails.getAuthorities()
    );

    //RETURN AUTHENTICATION OBJECT
    return authentication;

  }

}
```

MyUserDetailsService.java

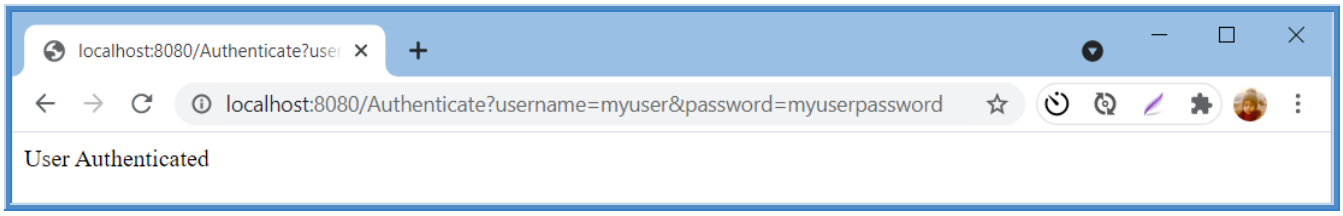
```java
package com.ivoronline.springboot_security_manual3.services;

import org.springframework.security.core.GrantedAuthority;
import org.springframework.security.core.authority.SimpleGrantedAuthority;
import org.springframework.security.core.userdetails.User;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.core.userdetails.UsernameNotFoundException;
import org.springframework.stereotype.Service;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;

@Service
public class MyUserDetailsService implements UserDetailsService {

  @Override
  public UserDetails loadUserByUsername(String enteredUsername) throws UsernameNotFoundException {

    //-----------------------------------------------------------------------------------------------
    //MOCK DB: <username, {"password", "ROLE"}>
    HashMap<String, String[]> accounts = new HashMap<>();
                              accounts.put("myuser" , new String[]{"myuserpassword" , "ROLE_USER" });
                              accounts.put("myadmin", new String[]{"myadminpassword", "ROLE_ADMIN"});
    //-----------------------------------------------------------------------------------------------

    //GET USER/ACCOUNT (From MOCK DB)
    String[] account = accounts.get(enteredUsername);

    //CHECK IF USER EXISTS
    if (account == null) { throw new UsernameNotFoundException(enteredUsername); }     //Bad credentials

    //GET PASSWORD
    String storedPassword = account[0];

    //GET AUTHORITIES
    List<GrantedAuthority> authorities = new ArrayList<GrantedAuthority>();
                           authorities.add(new SimpleGrantedAuthority(account[1]));

    //CREATE USER DETAILS OBJECT
    UserDetails userDetails = new User(enteredUsername, storedPassword, authorities);

    //RETURN USER DETAILS OBJECT
    return userDetails ;

  }

}
```

MyController.java

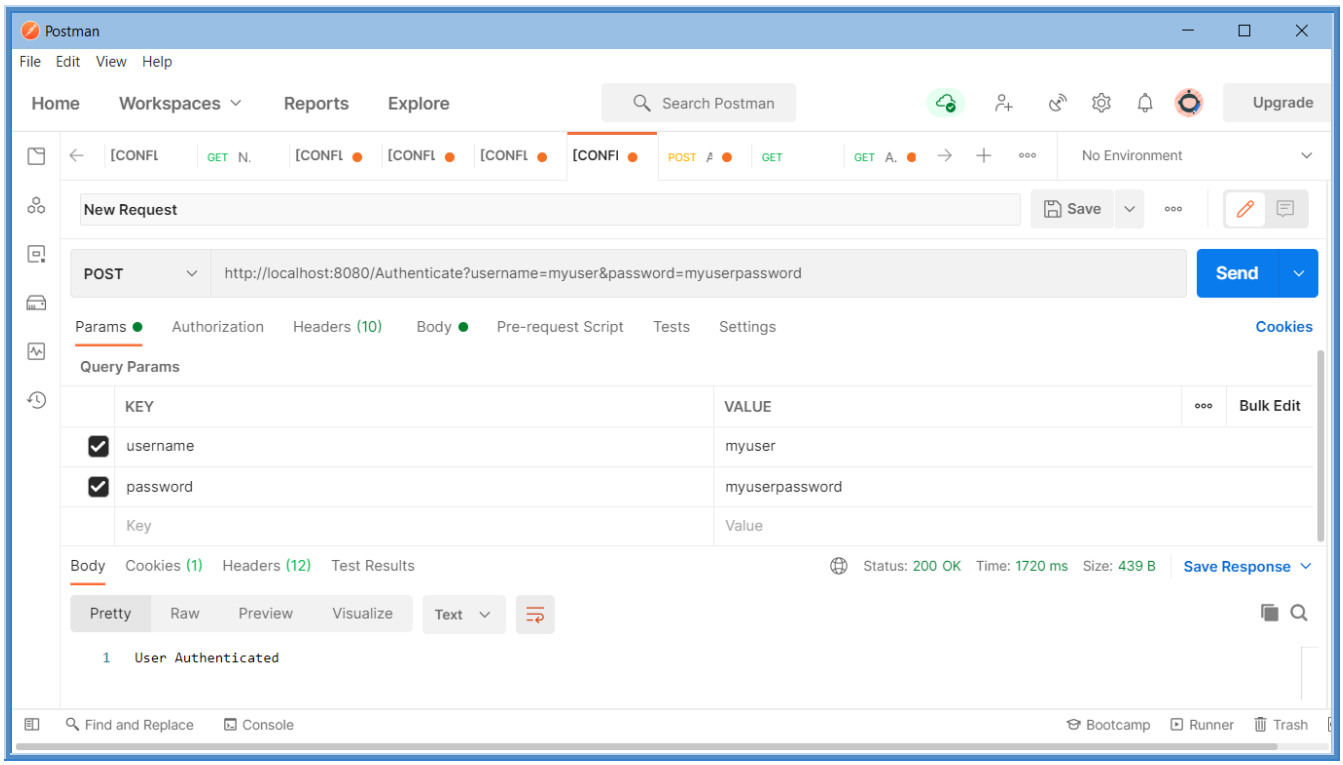
```
package com.ivoronline.springboot_security_manual3.controllers;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.access.annotation.Secured;
import org.springframework.security.authentication.AuthenticationManager;
import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;
import org.springframework.security.core.Authentication;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class MyController {

  @Autowired AuthenticationManager authenticationManager;

  //=========================================================================
  // AUTHENTICATE
  //=========================================================================
  @RequestMapping("/Authenticate")
  public String authenticate(@RequestParam String username, @RequestParam String password) {

    //CREATE AUTHENTICATION OBJECT (with Entered Username & Password)
    Authentication authentication = new UsernamePasswordAuthenticationToken(username, password);

    //GET    AUTHENTICATION OBJECT (with Authorities)
    authentication = authenticationManager.authenticate(authentication);

    //CHECK AUTHENTICATION
    if(authentication == null) { return "User NOT Authenticated"; }

    //STORE  AUTHENTICATION OBJECT (into Context)
    SecurityContextHolder.getContext().setAuthentication(authentication);

    //RETURN SOMETHING
    return "User Authenticated";

  }

  //=========================================================================
  // HELLO
  //=========================================================================
  @Secured("ROLE_USER")
  @RequestMapping("/Hello")
  public String hello() {
    return "Hello from Controller";
  }

}
```

Results

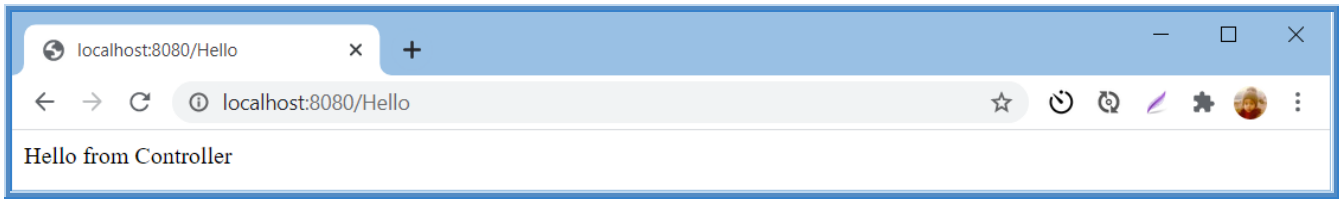
http://localhost:8080/Authenticate?username=myuser&password=myuserpassword *(GET)*



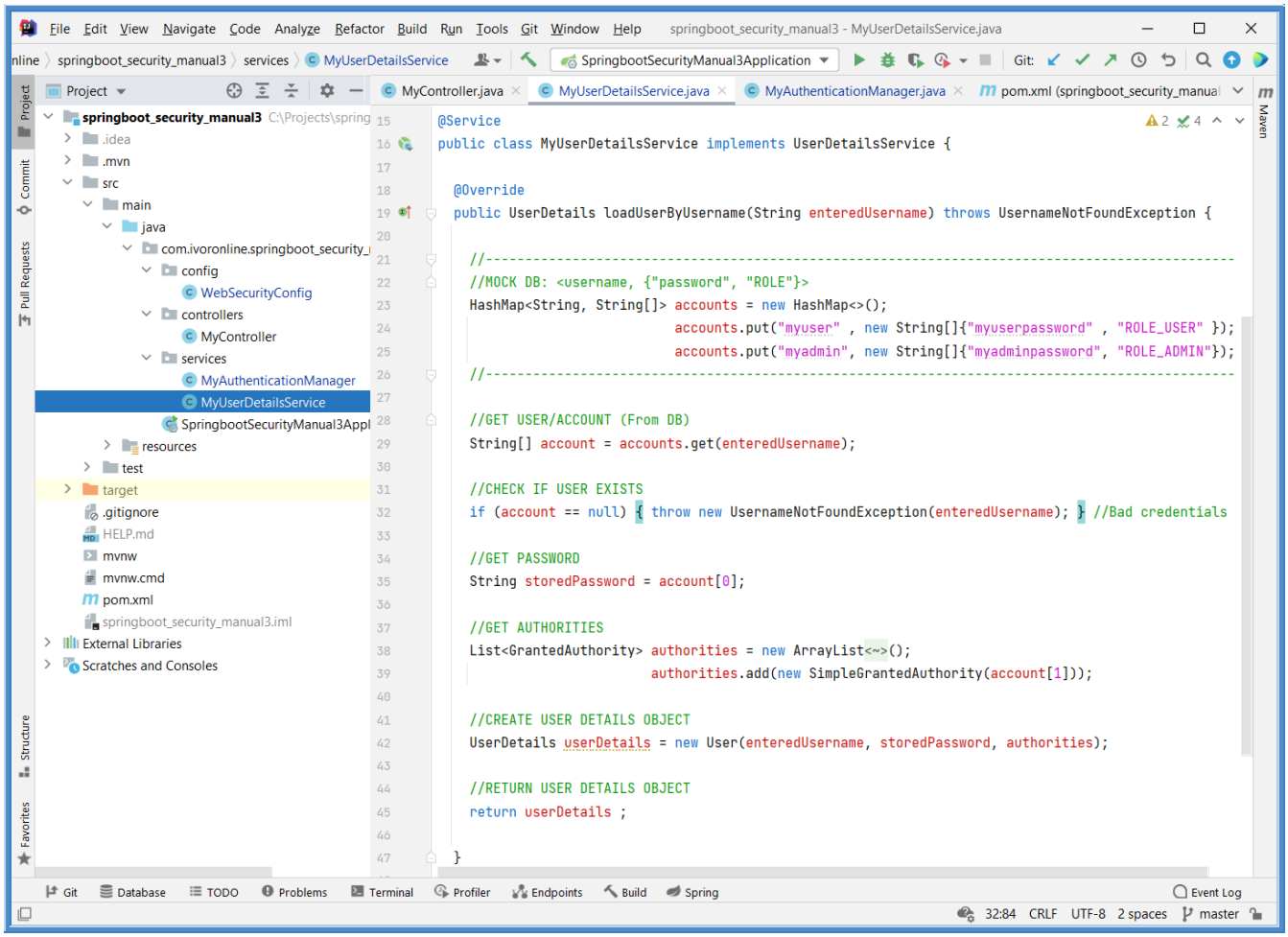
http://localhost:8080/Authenticate?username=myuser&password=myuserpassword *(POST)*



http://localhost:8080/Hello *(accessible only after authentication)*



Application Structure



pom.xml

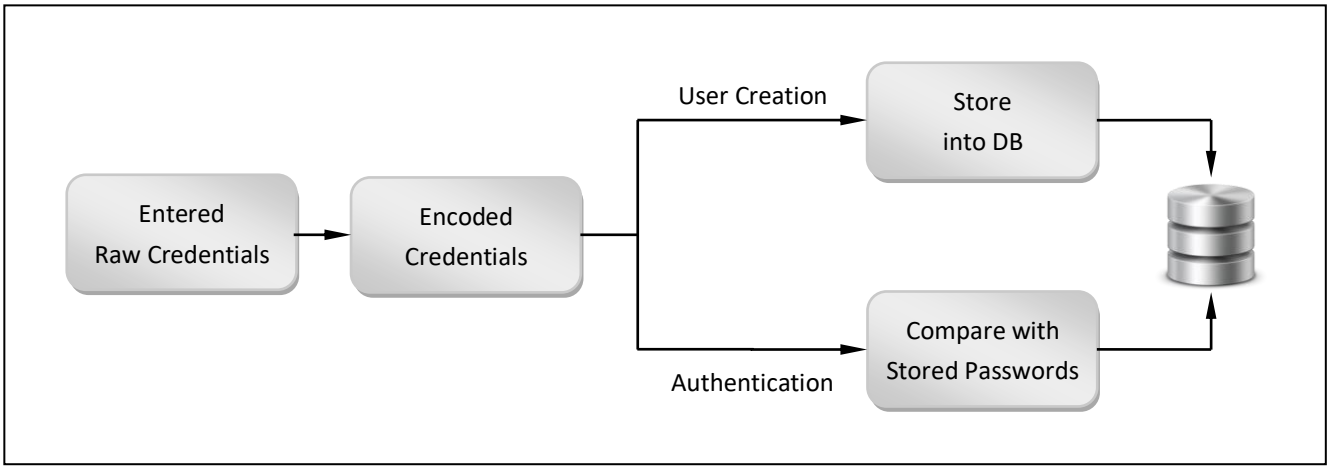
```
<dependencies>

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

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

  <dependency>
    <groupId>org.projectlombok</groupId>
    <artifactId>lombok</artifactId>
    <optional>true</optional>
  </dependency>

</dependencies>
```

2.5 Password Encoders

Info

- Password Encoders (Hash Algorithms) are used to encode **Entered Raw Password**
  - while **creating new User** (before storing them into DB to protect them if somebody gets access to DB)
  - during **Authentication** (so that Entered Encoded Password could be compared with Stored Encoded Passwords)

Password Encoders



Encoding Process

- When User wants to Login/Authenticate
  - he will Enter Raw Password
  - then Spring uses PasswordEncoder specified inside WebSecurityConfig to encode Entered Raw Password
  - and match it against Stored Encoded Password from DB
- Comparison is done by using matches() Method to compare Passwords by
  - taking Entered Raw Password
  - applying encoding algorithm (to encode Entered Raw Password)
  - calling matches() Method to compare result with Stored Encoded Password

Compare Passwords

```
if(passwordEncoder.matches(password, encodedPassword1)) { System.out.println("Passwords match");       }
else                                                     { System.out.println("Passwords don't match"); }
```

2.5.1 No Operation

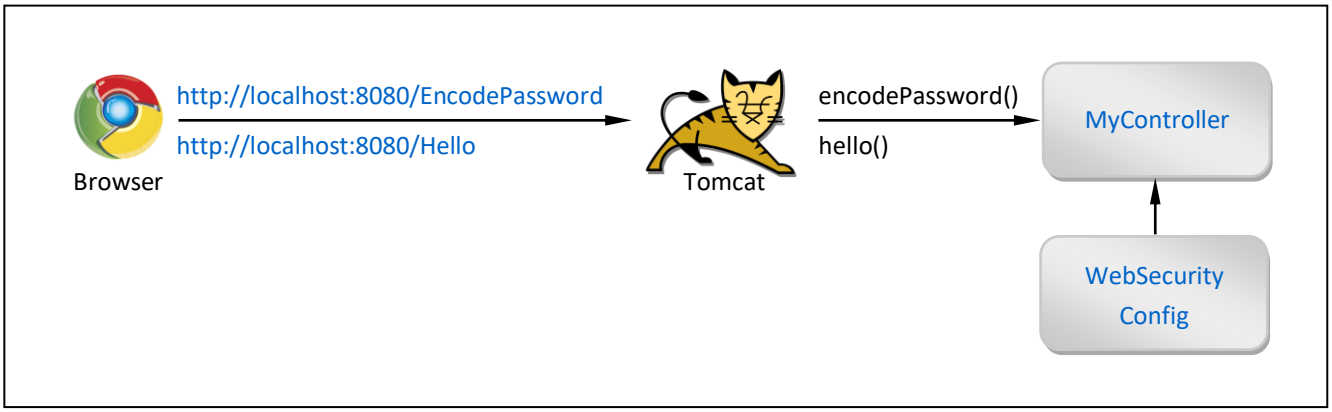
Info [G]

- No Operation Password Encoder doesn't actually encode Password at all.
- Instead it leaves it as it is in its original raw String format.
- But when you have Passwords that are not encoded you need to specify this encoder so that Spring would know how to properly read Stored Passwords (that is without decoding it).

Example

- In this tutorial User is defined inside application.properties and Password is not encoded.
  Therefore we are using No Operation Password Encoder to indicate that Stored Password is not Encoded.
  But "/EncodePassword" Endpoint is still included for easier comparison with other Password Encoders.
- Inside the Controller we have added "/EncodePassword" Endpoint which you can use to encode other Passwords.
  Inside WebSecurityConfig.java we have allowed Anonymous Access to this Endpoint.
  If you want to use another password
  - Start Application
    - call Endpoint http://localhost:8080/EncodePassword?password=**anotherpassword**
    - copy result into application.properties under spring.security.user.password
  - Restart Application
    - try to access http://localhost:8080/Hello
    - in the Login Form type **anotherpassword**

Application Schema *[Results]*



Spring Boot Starters

GROUP	DEPENDENCY	DESCRIPTION
Web	Spring Web	Enables @Controller, @RequestMapping and Tomcat Server
Security	Spring Security	Enables Spring Security

Procedure

- Create Project: springbott_security_passwordencoders_ldap (add Spring Boot Starters from the table)
- Edit File: application.properties (add Role, User, Password)
- Create Package: controllers (inside main package)
  - Create Class: MyController.java (inside package controllers)
- Create Package: config (inside main package)
  - Create Class: WebSecurityConfig.java (inside package config)

application.properties

```
# SECURITY
spring.security.user.name     = myuser
spring.security.user.password = mypassword
spring.security.user.roles    = USER
```

MyController.java

```
package com.ivoronline.springbott_security_passwordencoders_noop.controllers;

import org.springframework.security.crypto.password.NoOpPasswordEncoder;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.ResponseBody;

@Controller
public class MyController {

  @ResponseBody
  @RequestMapping("/EncodePassword")
  public String encodePassword(@RequestParam String password) {

    //GET PASSWORD ENCODER
    PasswordEncoder passwordEncoder = NoOpPasswordEncoder.getInstance();

    //ENCODE PASSWORD
    String encodedPassword = passwordEncoder.encode(password);

    //RETURN ENCODED PASSWORD
    return encodedPassword;

  }

  @ResponseBody
  @RequestMapping("/Hello")
  public String hello() {
    return "Hello from Controller";
  }

}
```

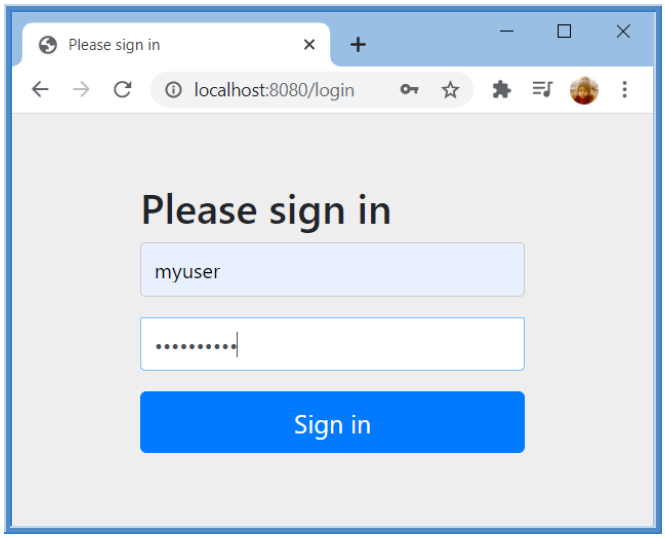
WebSecurityConfig.java

```
package com.ivoronline.springbott_security_passwordencoders_noop.config;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
import org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;
import org.springframework.security.crypto.password.NoOpPasswordEncoder;
import org.springframework.security.crypto.password.PasswordEncoder;

@Configuration
@EnableWebSecurity
public class WebSecurityConfig extends WebSecurityConfigurerAdapter {

  //=========================================================================
  // PASSWORD ENCODER
  //=========================================================================
  @Bean
  PasswordEncoder passwordEncoder() {
    return NoOpPasswordEncoder.getInstance();
  }

  //=========================================================================
  // CONFIGURE
  //=========================================================================
  @Override
  protected void configure(HttpSecurity httpSecurity) throws Exception {
    httpSecurity.authorizeRequests().antMatchers("/EncodePassword").permitAll(); //Anonymouse Access
    httpSecurity.authorizeRequests().anyRequest().authenticated();               //Authenticated Access
    httpSecurity.formLogin();                                                    //Default Login Form

  }

}
```

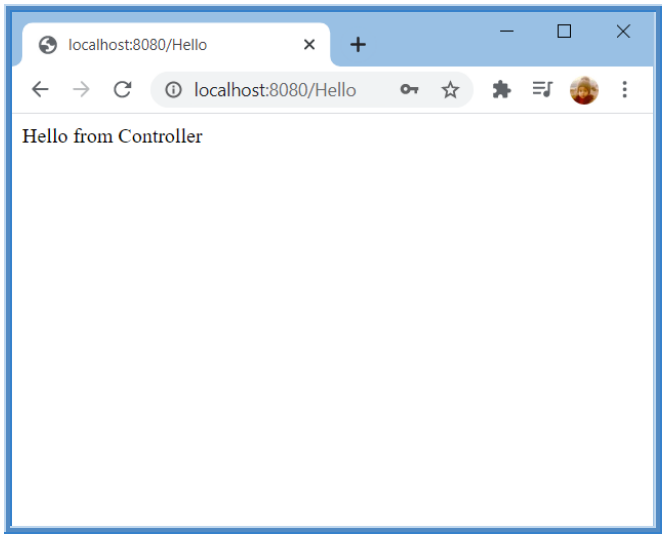
Results

- http://localhost:8080/Hello
- You get redirected to http://localhost:8080/login
  - Username: **myuser**
  - Password: **mypassword**
  - Sign in
- You get redirected to http://localhost:8080/Hello

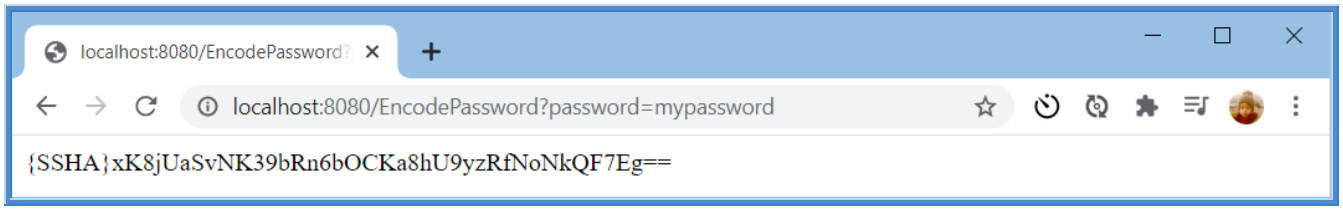
http://localhost:8080/login



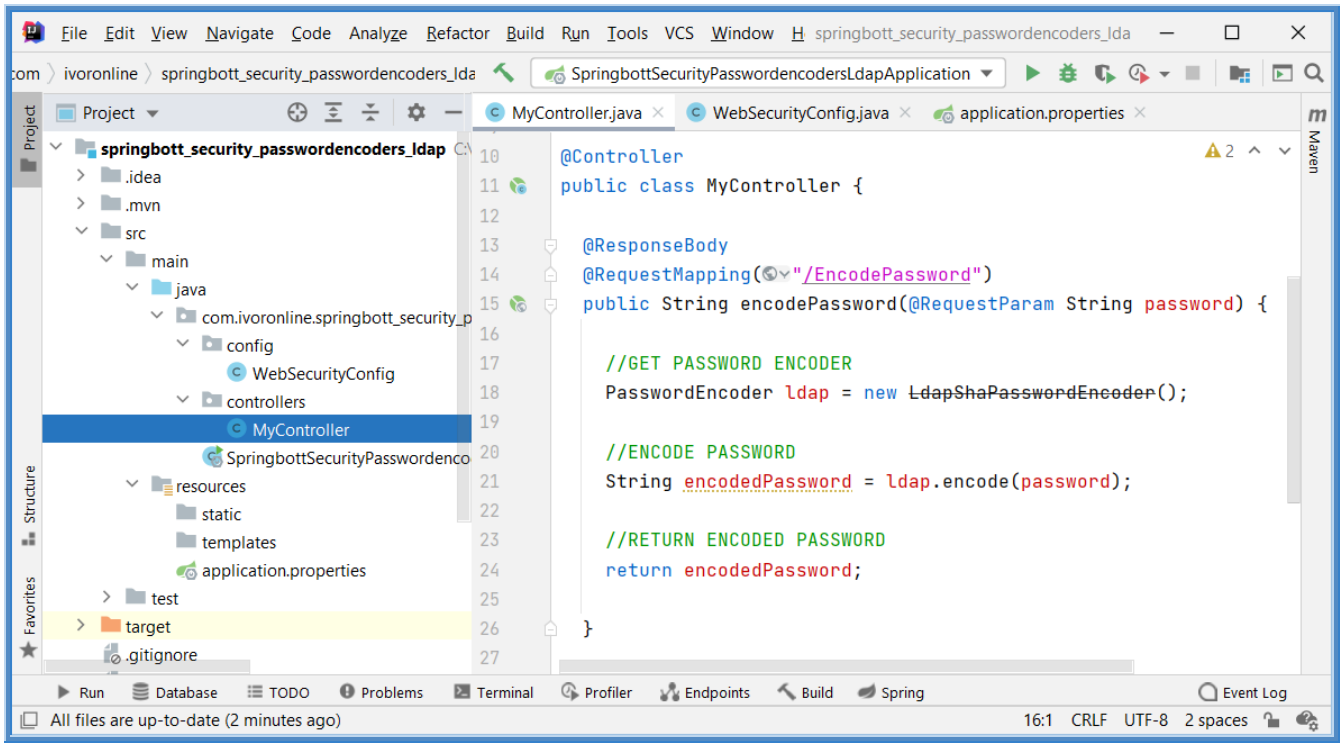
http://localhost:8080/Hello



*http://localhost:8080/EncodePassword?password=**mypassword*** *(if you want to encode another Password)*



Application Structure



pom.xml

```
<dependencies>

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

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

</dependencies>
```

2.5.2 LDAP

Info [G]

- LDAP Encoder uses
  - **random salt** to generate different Password Hash every time
  - matches() Method to compare Stored Encoded Password with the Entered Raw Password
- There are two Maven dependencies that you can use to work with LDAP Encoder
  - spring-security-core will not add Login Form
  - spring-boot-starter-security will add Login Form (this one is added when you select Security Spring Boot Starter)

pom.xml *(this one will not add Login Form)*

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

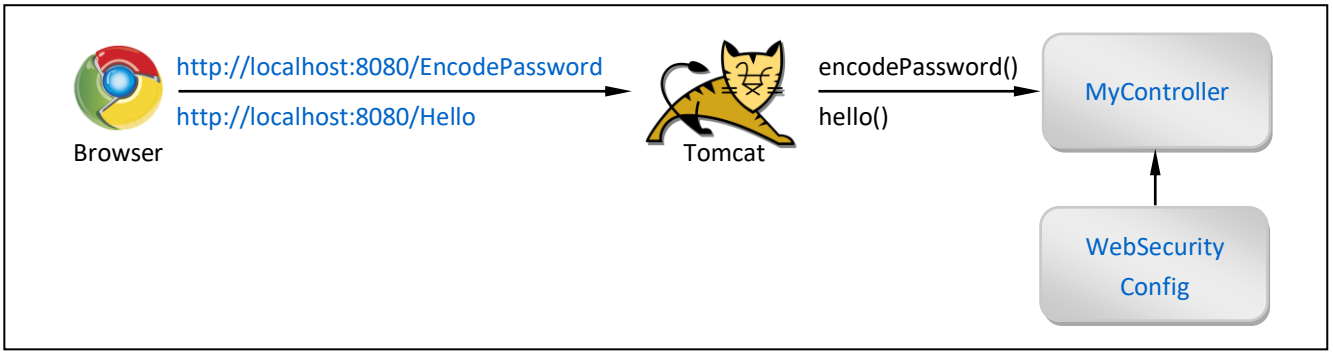
pom.xml *(this one will add Login Form)*

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

Example

- In this tutorial User is defined inside application.properties.
  But instead of providing Password in raw format we will provide **LDAP** Encoded Password.
  mypassword gets encoded into **{SSHA}xK8jUaSvNK39bRn6bOCKa8hU9yzRfNoNkQF7Eg==**.
- Inside the Controller we have added "/EncodePassword" Endpoint which you can use to encode other Passwords.
  Inside WebSecurityConfig.java we have allowed Anonymous Access to this Endpoint. If you want to use another password
  - Start Application
    - call Endpoint http://localhost:8080/EncodePassword?password=**anotherpassword**
    - copy result into application.properties under spring.security.user.password
  - Restart Application
    - try to access http://localhost:8080/Hello
    - in the Login Form type **anotherpassword**

Application Schema *[Results]*



Spring Boot Starters

GROUP	DEPENDENCY	DESCRIPTION
Web	Spring Web	Enables @Controller, @RequestMapping and Tomcat Server
Security	Spring Security	Enables Spring Security

Procedure

- Create Project: springbott_security_passwordencoders_ldap (add Spring Boot Starters from the table)
- Edit File: application.properties (add Role, User, Password)
- Create Package: controllers (inside main package)
  - Create Class: MyController.java (inside package controllers)
- Create Package: config (inside main package)
  - Create Class: WebSecurityConfig.java (inside package config)

application.properties

```
# SECURITY
spring.security.user.name     = myuser
spring.security.user.password = {SSHA}xK8jUaSvNK39bRn6bOCKa8hU9yzRfNoNkQF7Eg==
spring.security.user.roles    = USER
```

MyController.java

```
package com.ivoronline.springbott_security_passwordencoders_ldap.controllers;

import org.springframework.security.crypto.password.LdapShaPasswordEncoder;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.ResponseBody;

@Controller
public class MyController {

  @ResponseBody
  @RequestMapping("/EncodePassword")
  public String encodePassword(@RequestParam String password) {

    //GET PASSWORD ENCODER
    PasswordEncoder passwordEncoder = new LdapShaPasswordEncoder();

    //ENCODE PASSWORD
    String encodedPassword = passwordEncoder.encode(password);

    //RETURN ENCODED PASSWORD
    return encodedPassword;

  }

  @ResponseBody
  @RequestMapping("/Hello")
  public String hello() {
    return "Hello from Controller";
  }

}
```

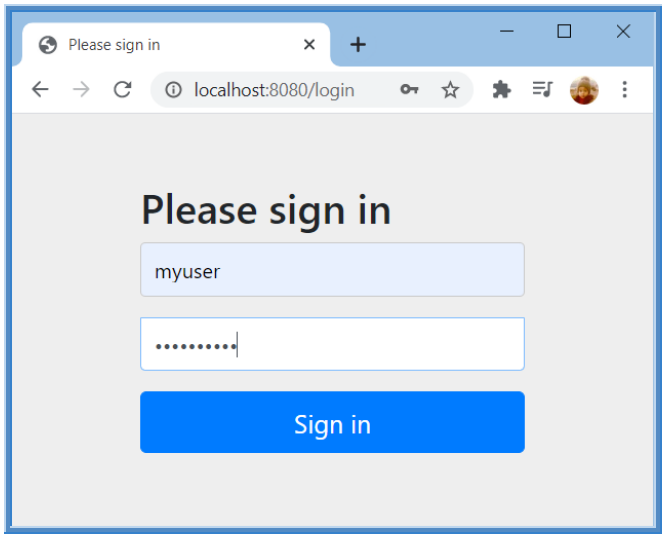
WebSecurityConfig.java

```
package com.ivoronline.springbott_security_passwordencoders_ldap.config;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
import org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;
import org.springframework.security.crypto.password.LdapShaPasswordEncoder;
import org.springframework.security.crypto.password.PasswordEncoder;

@Configuration
@EnableWebSecurity
public class WebSecurityConfig extends WebSecurityConfigurerAdapter {

  //=========================================================================
  // PASSWORD ENCODER
  //=========================================================================
  @Bean
  PasswordEncoder passwordEncoder() {
    return new LdapShaPasswordEncoder();
  }

  //=========================================================================
  // CONFIGURE
  //=========================================================================
  @Override
  protected void configure(HttpSecurity httpSecurity) throws Exception {
    httpSecurity.authorizeRequests().antMatchers("/EncodePassword").permitAll(); //Anonymouse Access
    httpSecurity.authorizeRequests().anyRequest().authenticated();               //Authenticated Access
    httpSecurity.formLogin();                                                    //Default Logn Form

  }

}
```

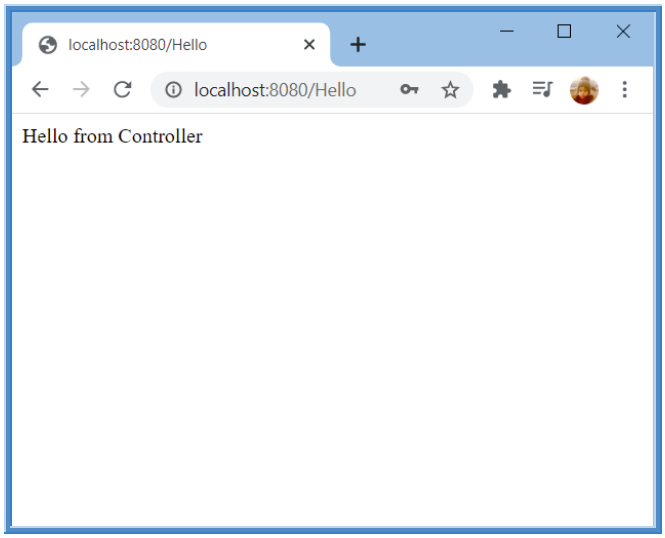
Results

- http://localhost:8080/Hello
- You get redirected to http://localhost:8080/login
  - Username: **myuser**
  - Password: **mypassword**
  - Sign in
- You get redirected to http://localhost:8080/Hello

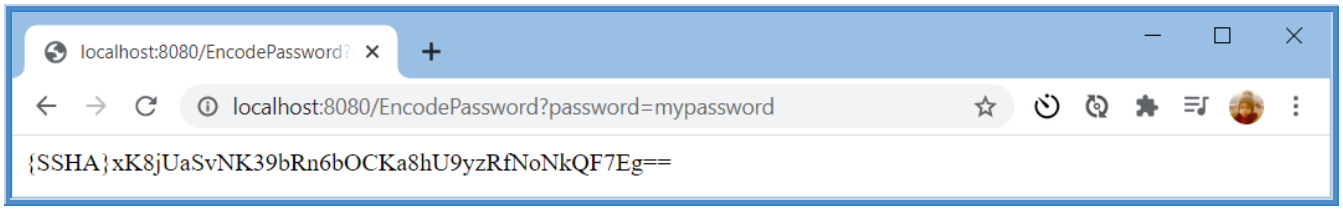
http://localhost:8080/login



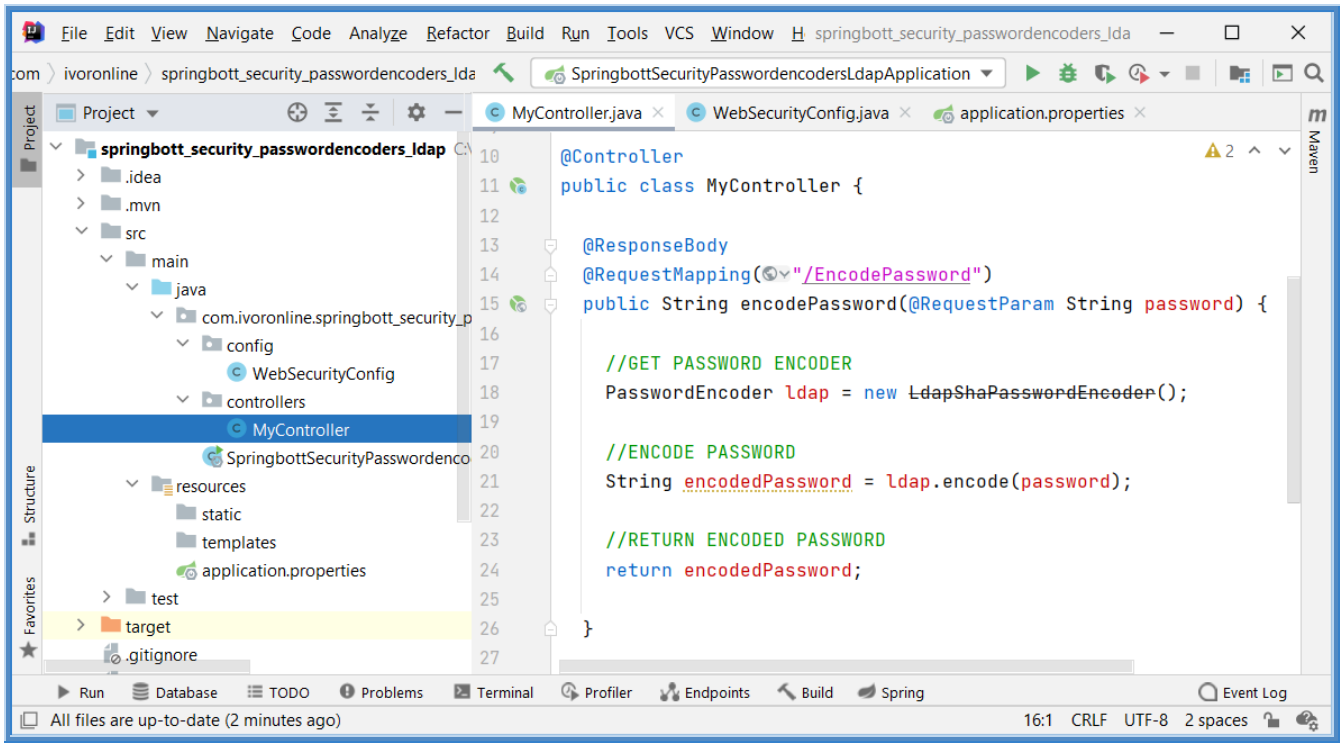
http://localhost:8080/Hello



*http://localhost:8080/EncodePassword?password=**mypassword*** *(if you want to encode another Password)*



Application Structure



pom.xml

```
<dependencies>

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

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

</dependencies>
```

3 Additional Terms

Info

- Following tutorials explain other terms related to Spring Boot Security like specific security implementations.

3.1 Remember Me

Info

- Following tutorials show how to use Remember Me Cookie to keep the User logged in indefinitely.
- That is even if Session Cookie expires or gets deleted.

3.1.1 Login Form - Default

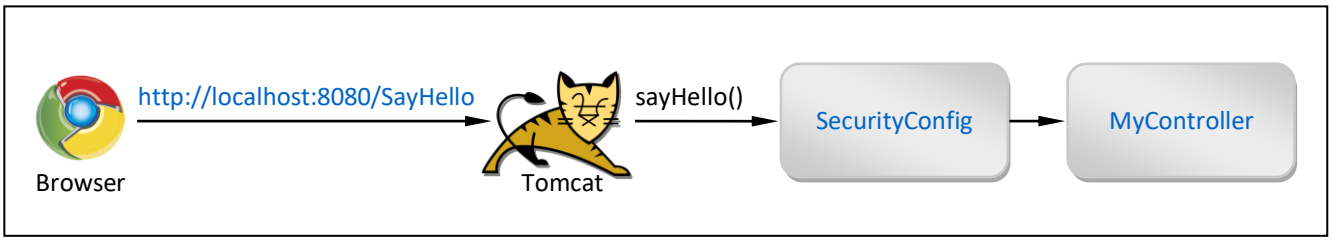
Info [G]

- This tutorial shows how to use Remember Me Cookie for Default Login Form.
  Once enabled Spring Boot will add Remember Me checkbox to Default Login Form.
- After successful Authentication through Default Login Form, Remember Me Cookie will be stored in the User's Browser.
  Remember Me Cookie will keep the User logged in permanently (even if Session Cookie has expired or was deleted).
  That way next time User sends HTTP Request it will not have to Login again.

SecurityConfig.java

```
@Configuration
@EnableWebSecurity
@RequiredArgsConstructor
public class SecurityConfig extends WebSecurityConfigurerAdapter {

  private final UserDetailsService userDetailsService;

  @Override
  protected void configure(HttpSecurity httpSecurity) throws Exception {

    //ENABLE REMEMBER ME COOKIE
    httpSecurity.rememberMe().key("something").userDetailsService(userDetailsService);
```

Application Schema *[Results]*



Spring Boot Starters

GROUP	DEPENDENCY	DESCRIPTION
Web	Spring Web	Enables @Controller, @RequestMapping and Tomcat Server.
Security	Spring Security	Enables Spring Security.
Developer Tools	Lombok	Enables @RequiredArgsConstructor

Procedure

- Create Project: springboot_security_remmberme (add Spring Boot Starters from the table)
- Edit File: application.properties (add Role, User, Password)
- Create Package: config (inside main package)
  - Create Class: SecurityConfig.java (inside package config)
- Create Package: controllers (inside main package)
  - Create Class: MyController.java (inside package controllers)

application.properties

```
# SECURITY
spring.security.user.name     = myuser
spring.security.user.password = mypassword
spring.security.user.roles    = USER
```

SecurityConfig.java

```
package com.ivoronline.springboot_security_remmberme.config;

import lombok.RequiredArgsConstructor;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
import org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;
import org.springframework.security.core.userdetails.UserDetailsService;

@Configuration
@EnableWebSecurity
@RequiredArgsConstructor
public class SecurityConfig extends WebSecurityConfigurerAdapter {

  private final UserDetailsService userDetailsService;

  @Override
  protected void configure(HttpSecurity httpSecurity) throws Exception {

    //ENABLE REMEMBER ME COOKIE
    httpSecurity.rememberMe().key("something").userDetailsService(userDetailsService);

    //DISABLE CSRF
    httpSecurity.csrf().disable();

    //SECURE EVERYTHING
    httpSecurity.authorizeRequests().anyRequest().authenticated();

    //DEFAULT LOGIN FORM
    httpSecurity.formLogin();

  }

}
```

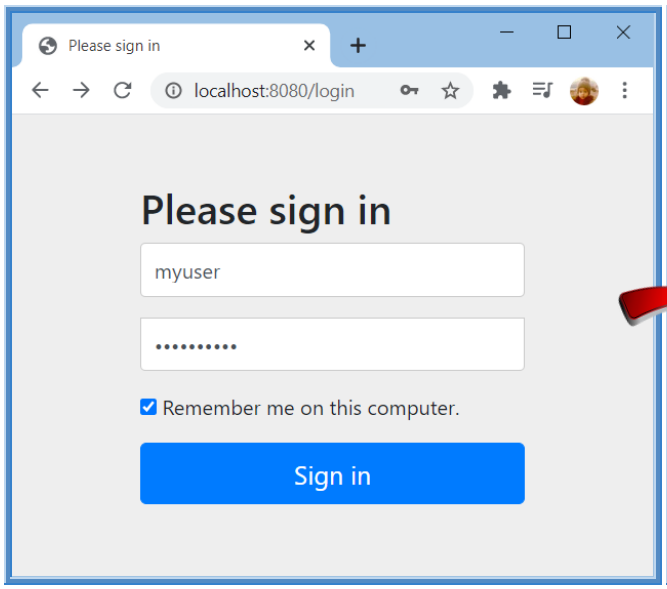
MyController.java

```
package com.ivoronline.springboot_security_remmberme.controllers;

import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.ResponseBody;

@Controller
public class MyController {

  @ResponseBody
  @RequestMapping("/SayHello")
  public String sayHello() {
    return "Hello from Controller";
  }

}
```

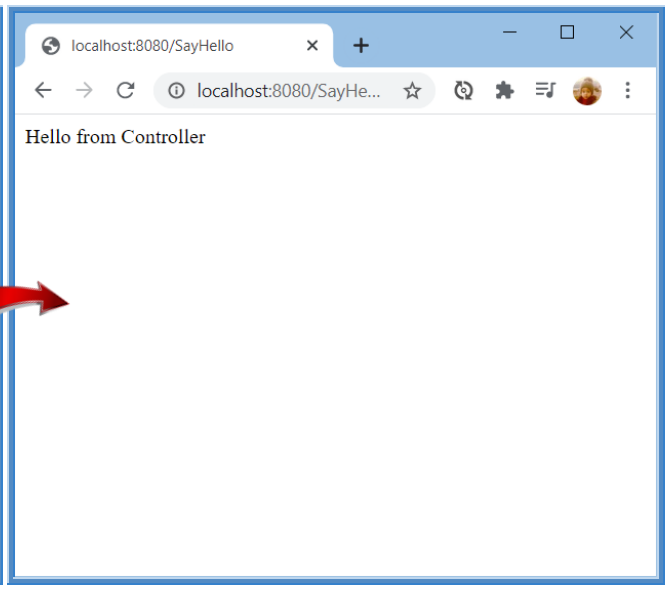
Results

- http://localhost:8080/SayHello
- You get redirected to http://localhost:8080/login
  - Username: myuser
  - Password: mypassword
  - Sign in
- You get redirected back to http://localhost:8080/SayHello
- Customize and control Google Chrome
  - More Tools
  - Developer Tools
  - Application
  - Cookies
  - http://localhost:8080
- Delete Cookie: JSESSIONID
  - http://localhost:8080/SayHello (you can access Page without logging in because remember-me Cookie is used)

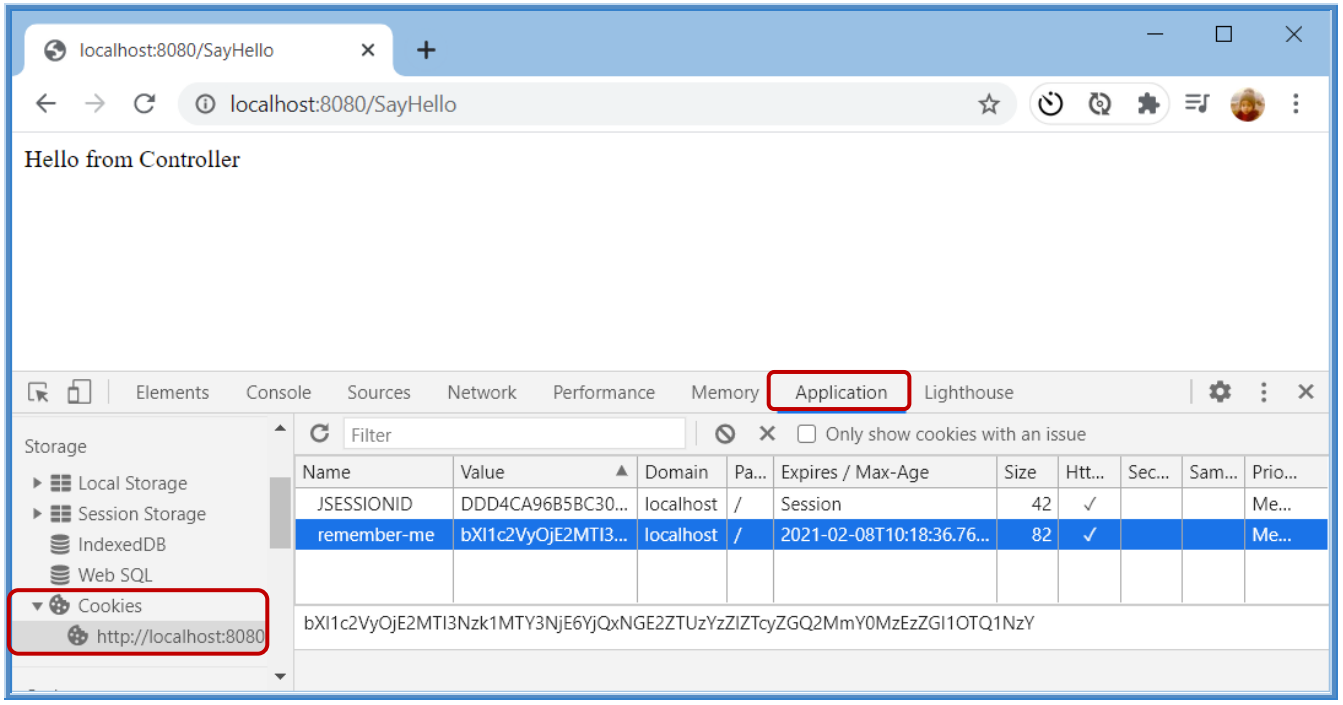
http://localhost:8080/login



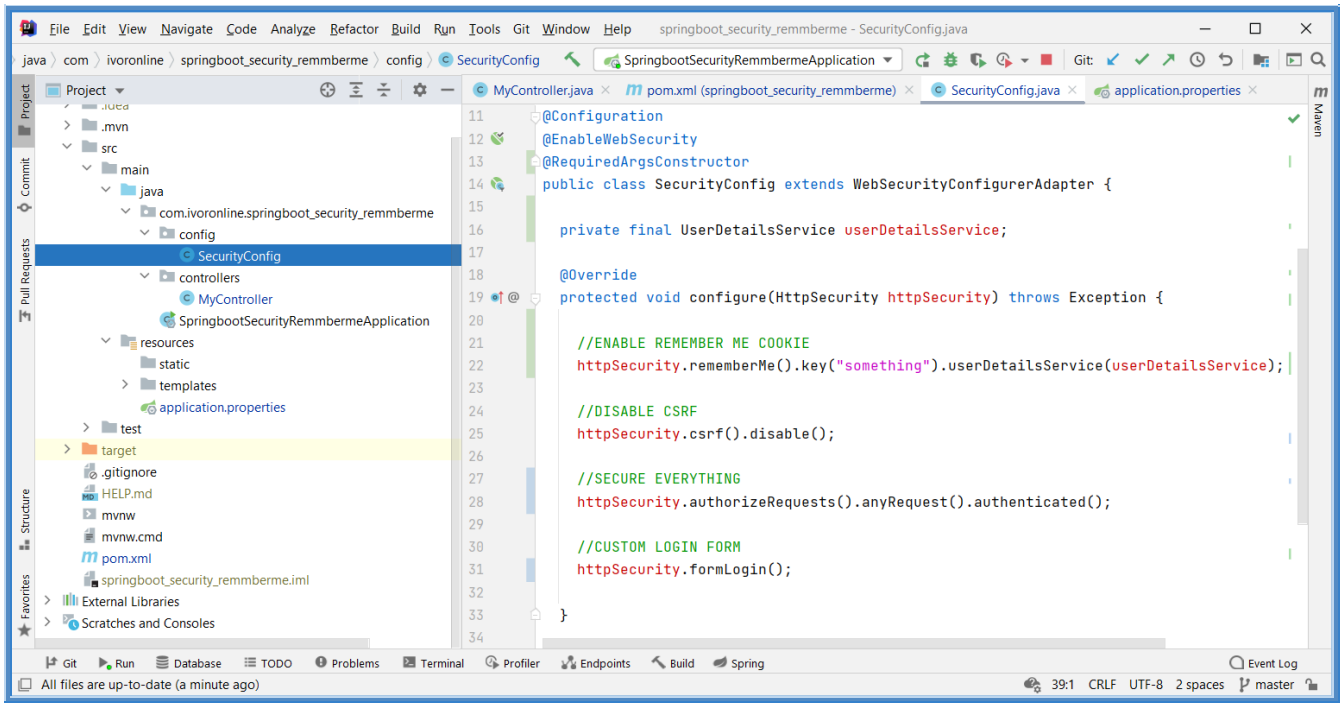
http://localhost:8080/SayHello



remember-me Cookie



Application Structure



pom.xml

```
<dependencies>

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

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

  <dependency>
    <groupId>org.projectlombok</groupId>
    <artifactId>lombok</artifactId>
    <optional>true</optional>
  </dependency>

</dependencies>
```

3.1.2 Login Form - Custom

Info

- This tutorial shows how to use Remember Me Cookie for Custom Login Form.
  This requires you to manually add additional checkbox to your Custom Login Form (as shown below).
- After successful Authentication through Custom Login Form, Remember Me Cookie will be stored in the User's Browser.
  Remember Me Cookie will keep the User logged in permanently (even if Session Cookie has expired or was deleted).
  That way next time User sends HTTP Request it will not have to Login again.

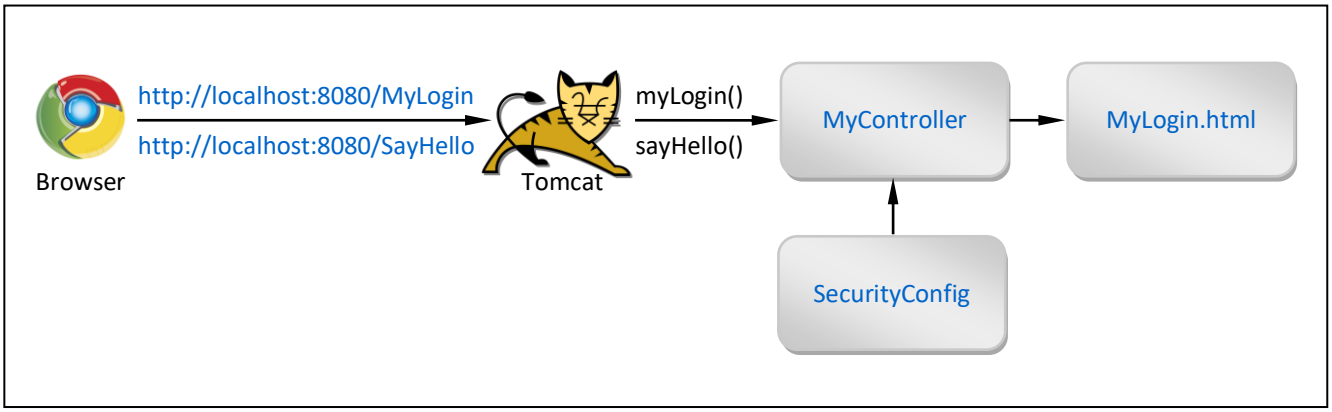
SecurityConfig.java

```
@RequiredArgsConstructor
private final UserDetailsService userDetailsService;
httpSecurity.rememberMe().key("something").userDetailsService(userDetailsService);
```

MyLogin.html

```
<input type="checkbox" name="remember-me"/>
```

Application Schema *[Results]*



Spring Boot Starters

GROUP	DEPENDENCY	DESCRIPTION
Web	Spring Web	Enables @Controller, @RequestMapping and Tomcat Server.
Security	Spring Security	Enables Spring Security.
Template Engines	Thyemeleaf	Enables Controller to return reference HTML Page index.html
Developer Tools	Lombok	Enables @Data which generate helper methods (setters, getters, ...)

Procedure

- Create Project: springboot_security_remmberme (add Spring Boot Starters from the table)
- Edit File: application.properties (add Role, User, Password)
- Create Package: config (inside main package)
  - Create Class: SecurityConfig.java (inside package config)
- Create Package: controllers (inside main package)
  - Create Class: MyController.java (inside package controllers)

application.properties

```
# SECURITY
spring.security.user.name     = myuser
spring.security.user.password = mypassword
spring.security.user.roles    = USER
```

SecurityConfig.java

```
package com.example.springboot_security_rememberme_customform.config;

import lombok.RequiredArgsConstructor;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
import org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;
import org.springframework.security.core.userdetails.UserDetailsService;

@Configuration
@EnableWebSecurity
@RequiredArgsConstructor
public class SecurityConfig extends WebSecurityConfigurerAdapter {

  private final UserDetailsService userDetailsService;

  @Override
  protected void configure(HttpSecurity httpSecurity) throws Exception {

    //ENABLE REMEMBER ME COOKIE
    httpSecurity.rememberMe().key("something").userDetailsService(userDetailsService);

    //DISABLE CSRF
    httpSecurity.csrf().disable();

    //SPECIFY ACCESS TO ENDPOINTS
    httpSecurity.authorizeRequests()
      .antMatchers("/SayHello").hasRole("USER");

    //CUSTOM LOGIN FORM
    httpSecurity.formLogin()
      .loginPage("/MyLogin")
      .loginProcessingUrl("/login");

  }

}
```

MyController.java

```
package com.example.springboot_security_rememberme_customform.controllers;

import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.ResponseBody;

@Controller
public class MyController {

  @ResponseBody
  @RequestMapping("/SayHello")
  public String sayHello() {
    return "Hello from Controller";
  }

  @RequestMapping("/MyLogin")
  public String myLogin() {
    return "MyLogin";
  }

}
```

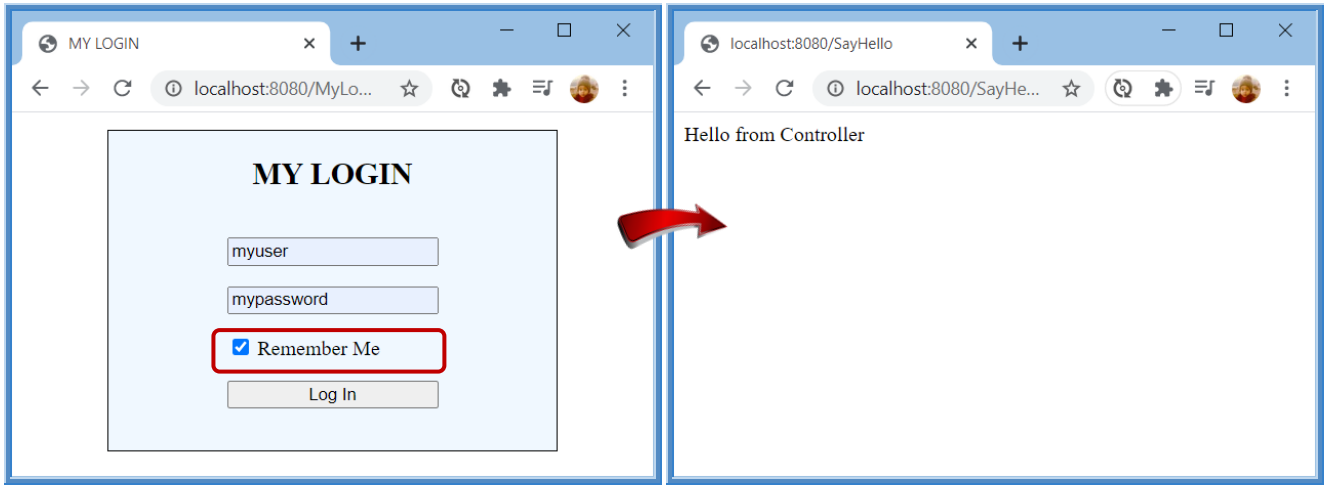
MyLogin.html

```
<html lang="en" xmlns:th="http://www.thymeleaf.org">
<title> MY LOGIN </title>

<style type="text/css">
    div { display:flex; flex-direction:column; align-items:center; border: solid 1pt; margin: 10pt 50pt;
background-color: aliceblue }
</style>

<div>
  <h2> MY LOGIN </h2>
  <form method="POST" action="login">
    <p> <input type="text"     name="username"  placeholder="username"            /> </p>
    <p> <input type="text"     name="password"  placeholder="password"            /> </p>
    <p> <input type="checkbox" name="remember-me"                                 /> </p>
    <p> <input type="submit"   name="addAuthor" value="Log In" style="width:100%" /> </p>
  </form>
</div>
```

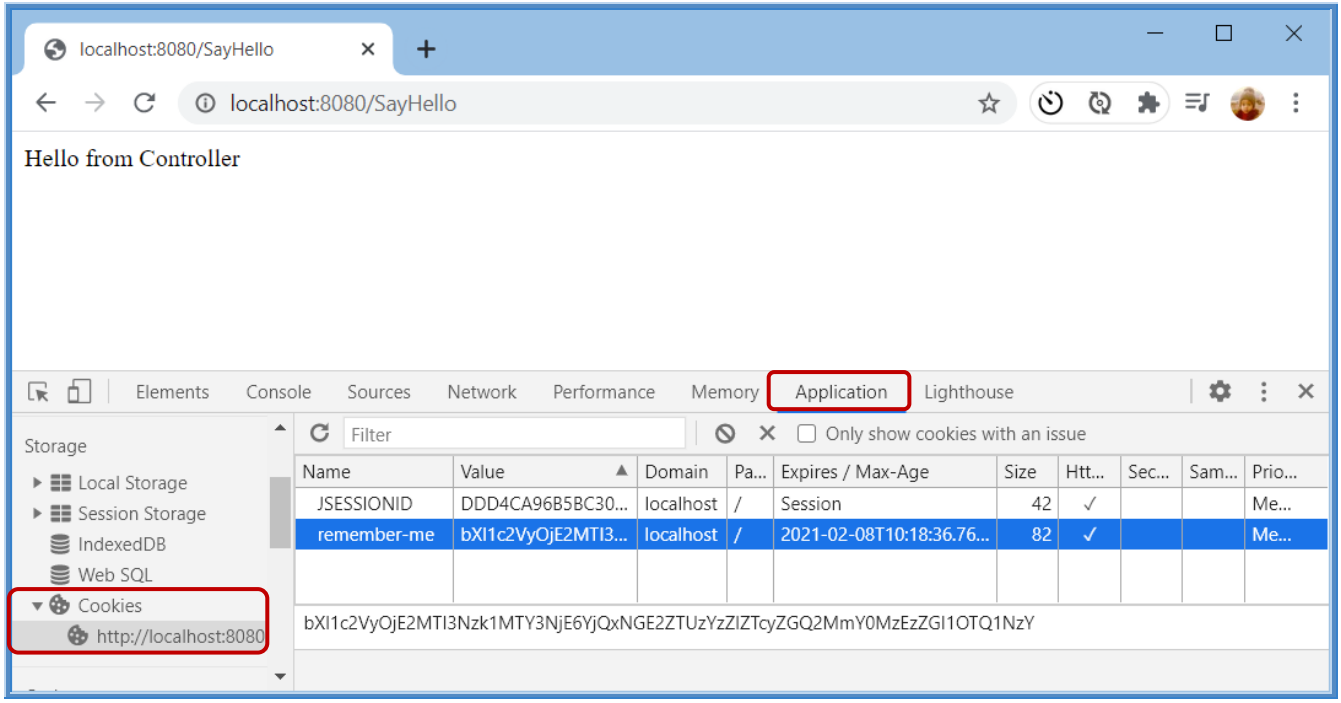
Results

- http://localhost:8080/SayHello
- You get redirected to http://localhost:8080/MyLogin
  - Username: myuser
  - Password: mypassword
  - **Remember Me: CHECK**
  - Sign in
- You get redirected back to http://localhost:8080/SayHello
- Customize and control Google Chrome
  - More Tools
  - Developer Tools
  - Application
  - Cookies
  - http://localhost:8080
- Delete Cookie: JSESSIONID
  - http://localhost:8080/SayHello (you can access Page without logging in because remember-me Cookie is used)

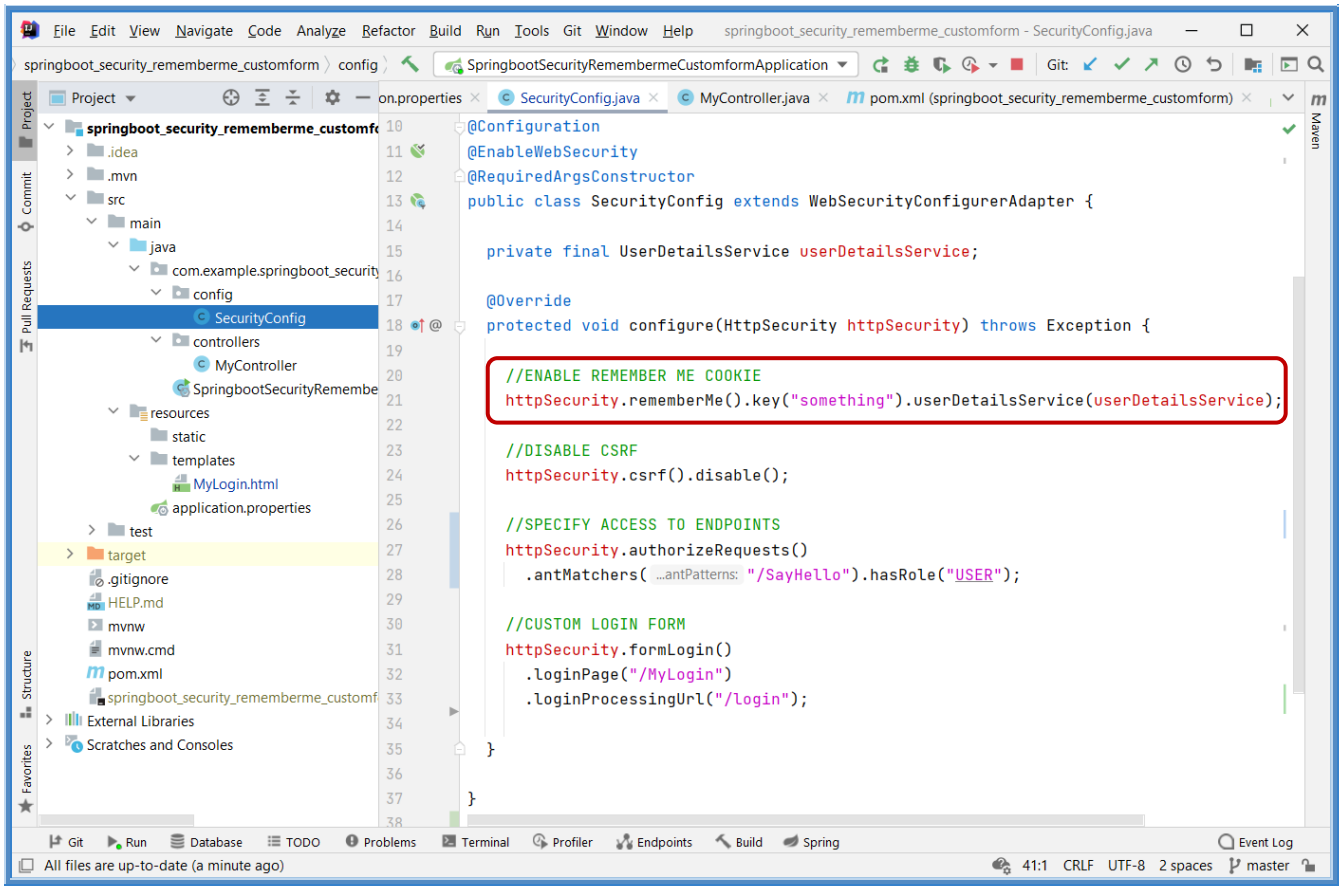
http://localhost:8080/login *http://localhost:8080/SayHello*



remember-me Cookie



Application Structure



pom.xml

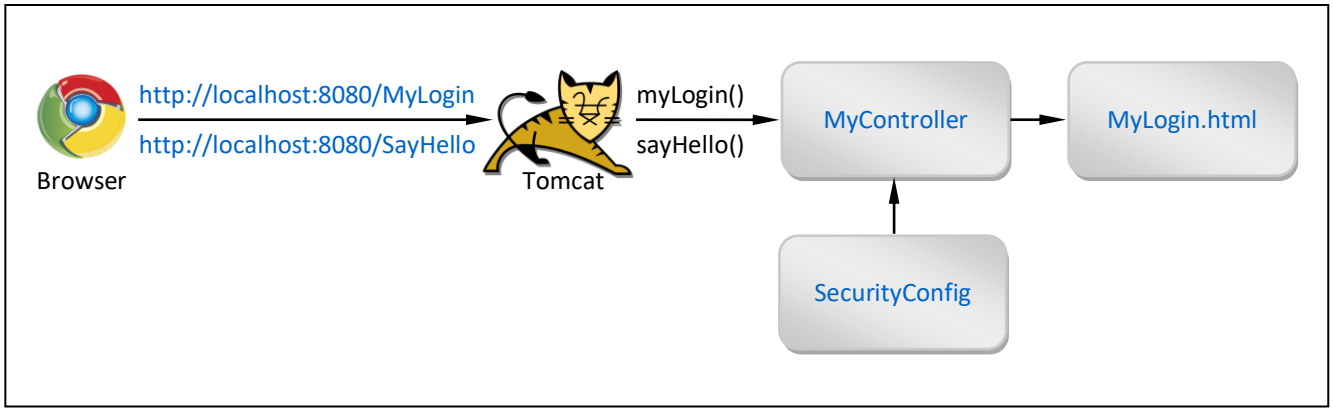
```
<dependencies>

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

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

  <dependency>
    <groupId>org.projectlombok</groupId>
    <artifactId>lombok</artifactId>
    <optional>true</optional>
  </dependency>

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

</dependencies>
```

3.1.3 Login Form - Default - DB - PostgreSQL

Info

- This tutorial shows how to use Remember Me Cookie for Custom Login Form.
  This requires you to manually add additional checkbox to your Custom Login Form (as shown below).
- After successful Authentication through Custom Login Form, Remember Me Cookie will be stored in the User's Browser.
  Remember Me Cookie will keep the User logged in permanently (even if Session Cookie has expired or was deleted).
  That way next time User sends HTTP Request it will not have to Login again.

Application Schema *[Results]*



Spring Boot Starters

GROUP	DEPENDENCY	DESCRIPTION
Web	Spring Web	Enables @Controller, @RequestMapping and Tomcat Server.
Security	Spring Security	Enables Spring Security.
SQL	Spring Data JPA	Enables @Entity and @Id
SQL	PostgreSQL Database	Enables Hibernate to work with PostgreSQL DB
Developer Tools	Lombok	Enables @Data which generate helper methods (setters, getters, ...)

Procedure

- Create Project: springboot_security_rememberme_db (add Spring Boot Starters from the table)
- Edit File: application.properties (add Role, User, Password)
- Create File: schema.sql (inside templates directory)
- Create Package: config (inside main package)
  - Create Class: SecurityConfig.java (inside package config)
  - Create Class: SecurityBeans.java (inside package config)
- Create Package: controllers (inside main package)
  - Create Class: MyController.java (inside package controllers)

application.properties

```
# SECURITY
spring.security.user.name             = myuser
spring.security.user.password         = mypassword
spring.security.user.roles            = USER

# POSTGRESQL DATABASE
spring.datasource.driver-class-name   = org.postgresql.Driver
spring.datasource.url                 = jdbc:postgresql://localhost:5432/postgres
spring.datasource.username            = postgres
spring.datasource.password            = letmein
spring.datasource.initialization-mode = always
```

schema.sql

```
create table persistent_logins (
  username  varchar(64) not null,
  series    varchar(64) primary key,
  token     varchar(64) not null,
  last_used timestamp   not null
);
```

SecurityBeans.java

```
package com.ivoronline.springboot_security_rememberme_db.config;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.web.authentication.rememberme.JdbcTokenRepositoryImpl;
import org.springframework.security.web.authentication.rememberme.PersistentTokenRepository;

import javax.sql.DataSource;

@Configuration
public class SecurityBeans {

  @Bean
  public PersistentTokenRepository persistentTokenRepository(DataSource dataSource) {
    JdbcTokenRepositoryImpl tokenRepository = new JdbcTokenRepositoryImpl();
                            tokenRepository.setDataSource(dataSource);
    return tokenRepository;

  }

}
```

SecurityConfig.java

```
package com.ivoronline.springboot_security_rememberme_db.config;

import lombok.RequiredArgsConstructor;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
import org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.web.authentication.rememberme.PersistentTokenRepository;

@Configuration
@EnableWebSecurity
@RequiredArgsConstructor
public class SecurityConfig extends WebSecurityConfigurerAdapter {

  @Autowired
  private final UserDetailsService        userDetailsService;
  private final PersistentTokenRepository persistentTokenRepository ;

  @Override
  protected void configure(HttpSecurity httpSecurity) throws Exception {

    //ENABLE REMEMBER ME COOKIE
    httpSecurity.rememberMe()
      .tokenRepository(persistentTokenRepository).userDetailsService(userDetailsService);

    //H2 CONSOLE
    httpSecurity.authorizeRequests(authorize -> { authorize.antMatchers("/h2-console/**").permitAll(); });
    httpSecurity.headers().frameOptions().sameOrigin();

    //DISABLE CSRF
    httpSecurity.csrf().disable();

    //SECURE EVERYTHING
    httpSecurity.authorizeRequests().anyRequest().authenticated();

    //DEfAULT LOGIN FORM
    httpSecurity.formLogin();

  }

}
```

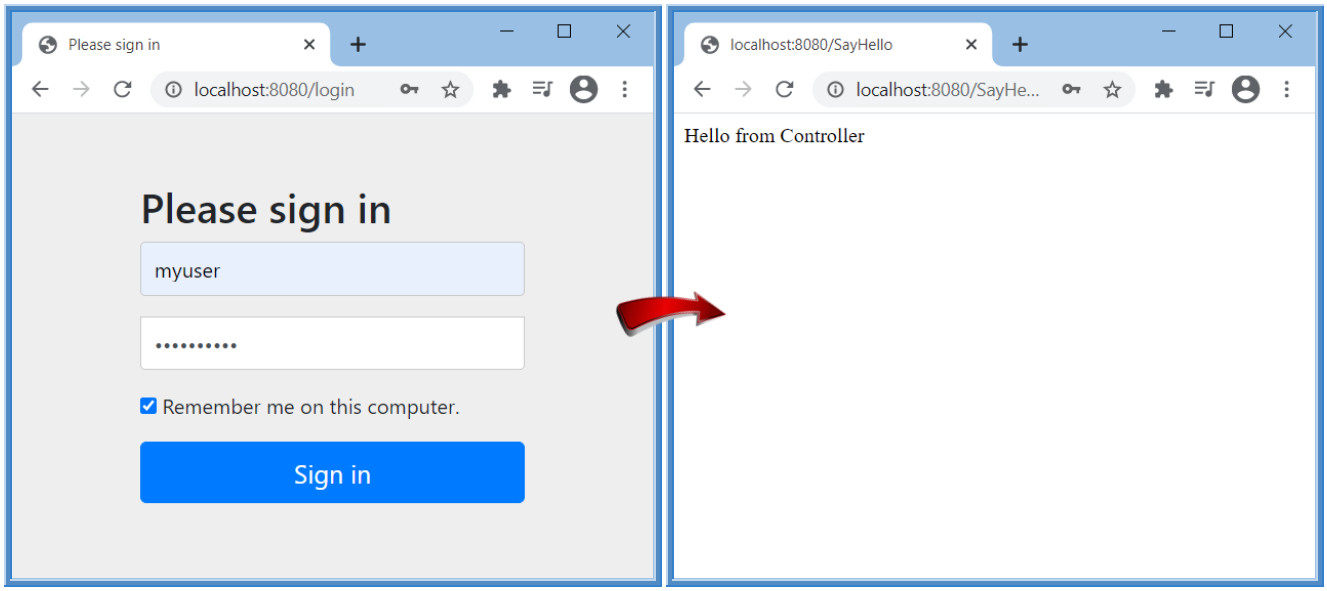
MyController.java

```
package com.ivoronline.springboot_security_rememberme_db.controllers;

import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.ResponseBody;

@Controller
public class MyController {

  @ResponseBody
  @RequestMapping("/SayHello")
  public String sayHello() {
    return "Hello from Controller";
  }
}
```

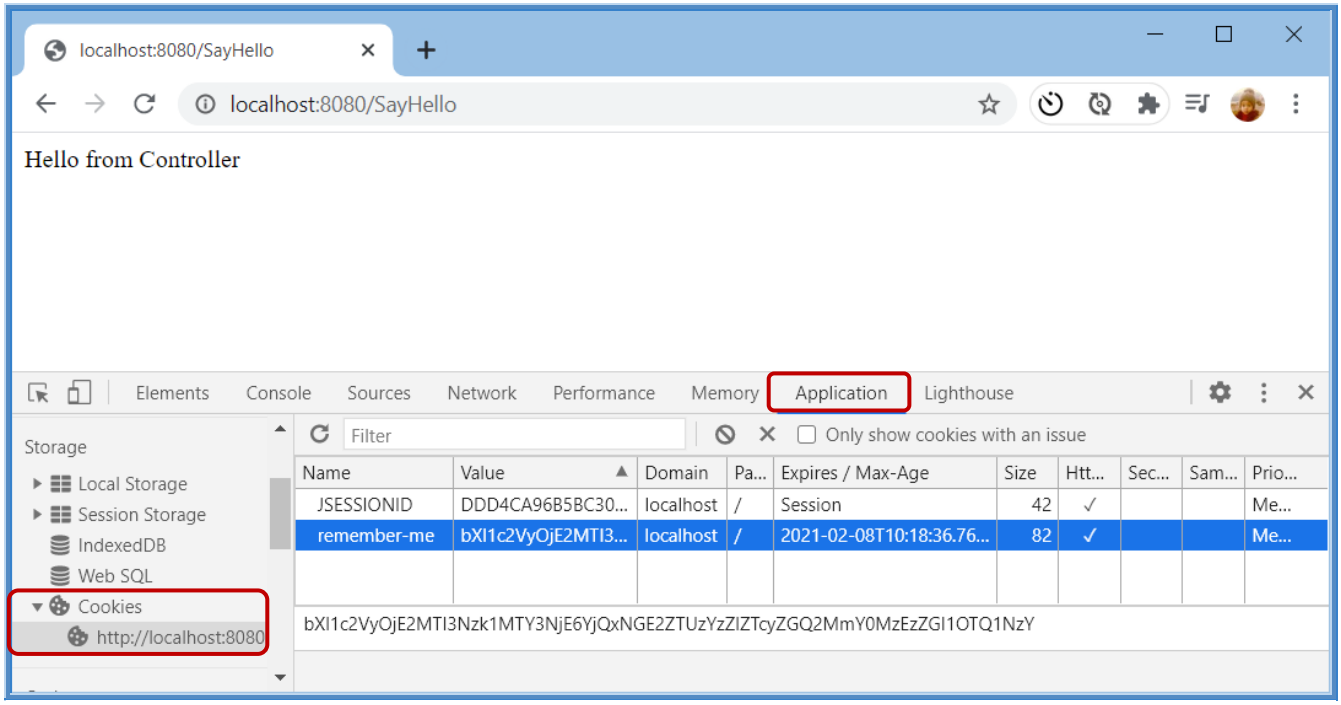
Results

- http://localhost:8080/SayHello
- You get redirected to http://localhost:8080/MyLogin
  - Username: myuser
  - Password: mypassword
  - **Remember Me: CHECK**
  - Sign in
- You get redirected back to http://localhost:8080/SayHello
- Customize and control Google Chrome
  - More Tools
  - Developer Tools
  - Application
  - Cookies
  - http://localhost:8080

http://localhost:8080/login *http://localhost:8080/SayHello*



remember-me Cookie



Persistent Token in DB Table: persistent_logins

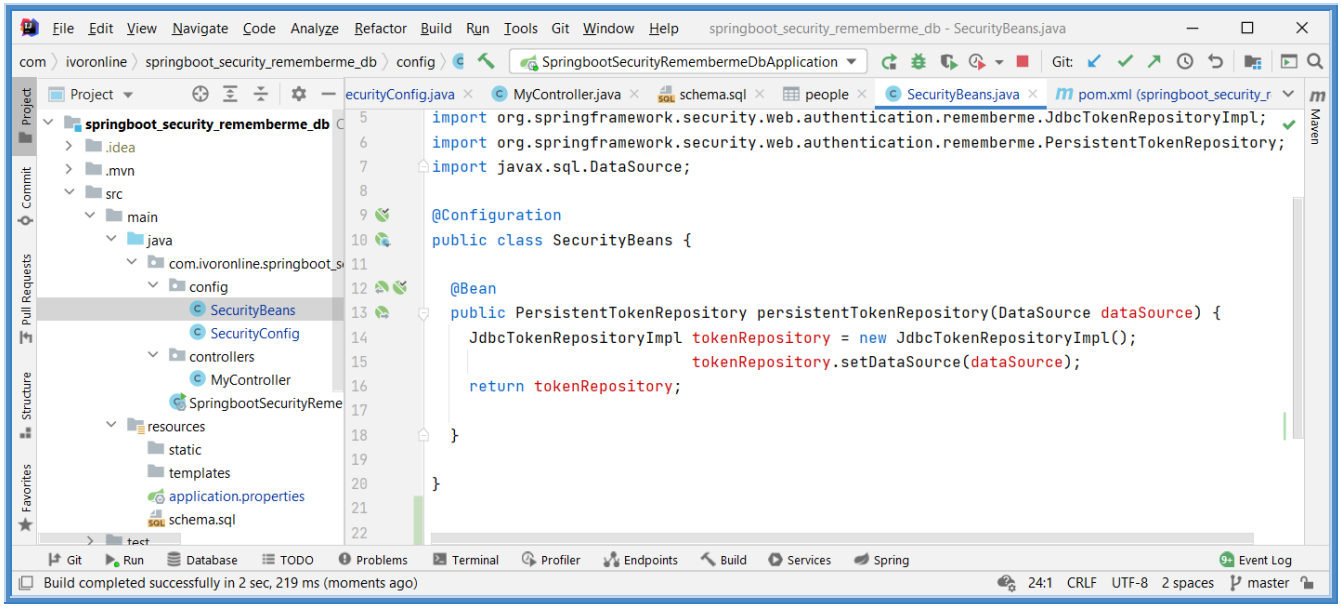
persistent_logins

1 row | Tx: Auto | DDL | DML | Comma-...d (CSV)

<Filter Criteria>

	username	series	token	last_used
1	myuser	esvmjF21+cB21PurJ31sTA==	BiZfMlI+4nXQt+DVXegUUQ==	2021-01-25 18:33:05.113000

Application Structure



pom.xml

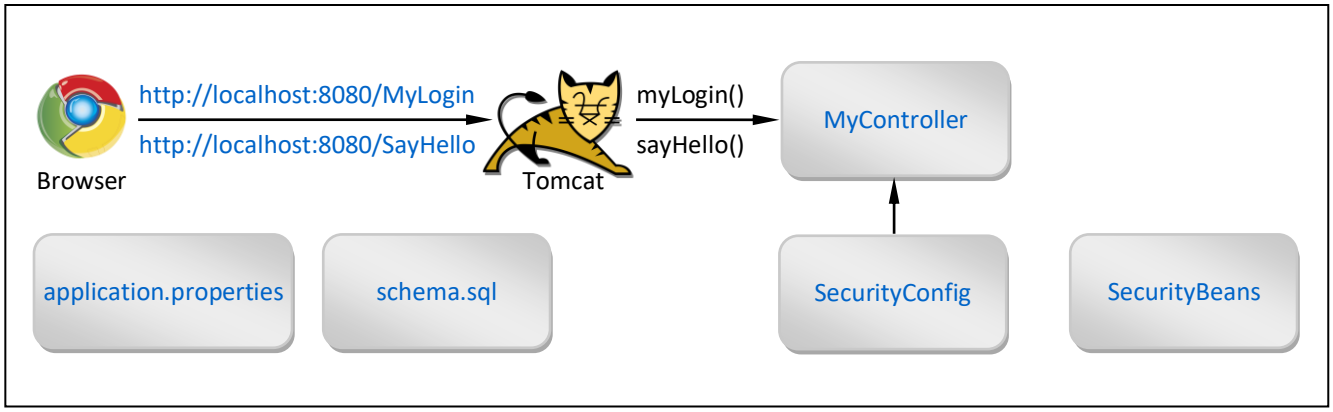
```
<dependencies>

  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</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-data-jpa</artifactId>
  </dependency>

  <dependency>
    <groupId>org.postgresql</groupId>
    <artifactId>postgresql</artifactId>
     <scope>runtime</scope>
  </dependency>

  <dependency>
    <groupId>org.projectlombok</groupId>
    <artifactId>lombok</artifactId>
    <optional>true</optional>
  </dependency>

</dependencies>
```

3.1.4 Login Form - Default - DB - H2

Info

- This tutorial shows how to use Remember Me Cookie for Custom Login Form.
  This requires you to manually add additional checkbox to your Custom Login Form (as shown below).
- After successful Authentication through Custom Login Form, Remember Me Cookie will be stored in the User's Browser.
  Remember Me Cookie will keep the User logged in permanently (even if Session Cookie has expired or was deleted).
  That way next time User sends HTTP Request it will not have to Login again.

Application Schema *[Results]*



Spring Boot Starters

GROUP	DEPENDENCY	DESCRIPTION
Web	Spring Web	Enables @Controller, @RequestMapping, Tomcat Server
Security	Spring Security	Enables Spring Security
SQL	Spring Data JPA	Enables @Entity and @Id
SQL	PostgreSQL Database	Enables Hibernate to work with PostgreSQL DB
Developer Tools	Lombok	Enables @Data which generate helper methods (setters, getters, ...)

Procedure

- Create Project: springboot_security_rememberme_db (add Spring Boot Starters from the table)
- Edit File: application.properties (add Role, User, Password)
- Create File: schema.sql (inside templates directory)
- Create Package: config (inside main package)
  - Create Class: SecurityConfig.java (inside package config)
  - Create Class: SecurityBeans.java (inside package config)
- Create Package: controllers (inside main package)
  - Create Class: MyController.java (inside package controllers)

application.properties

```
# SECURITY
spring.security.user.name             = myuser
spring.security.user.password         = mypassword
spring.security.user.roles            = USER

# H2 DATABASE
spring.h2.console.enabled = true
spring.datasource.url                 = jdbc:h2:mem:testdb
spring.datasource.initialization-mode = always
```

schema.sql

```
create table persistent_logins (
  username  varchar(64) not null,
  series    varchar(64) primary key,
  token     varchar(64) not null,
  last_used timestamp   not null
);
```

SecurityBeans.java

```
package com.ivoronline.springboot_security_rememberme_db.config;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.web.authentication.rememberme.JdbcTokenRepositoryImpl;
import org.springframework.security.web.authentication.rememberme.PersistentTokenRepository;

import javax.sql.DataSource;

@Configuration
public class SecurityBeans {

  @Bean
  public PersistentTokenRepository persistentTokenRepository(DataSource dataSource) {
    JdbcTokenRepositoryImpl tokenRepository = new JdbcTokenRepositoryImpl();
                            tokenRepository.setDataSource(dataSource);
    return tokenRepository;

  }

}
```

SecurityConfig.java

```
package com.ivoronline.springboot_security_rememberme_db.config;

import lombok.RequiredArgsConstructor;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
import org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.web.authentication.rememberme.PersistentTokenRepository;

@Configuration
@EnableWebSecurity
@RequiredArgsConstructor
public class SecurityConfig extends WebSecurityConfigurerAdapter {

  @Autowired
  private final UserDetailsService        userDetailsService;
  private final PersistentTokenRepository persistentTokenRepository ;

  @Override
  protected void configure(HttpSecurity httpSecurity) throws Exception {

    //ENABLE REMEMBER ME COOKIE
    httpSecurity.rememberMe()
      .tokenRepository(persistentTokenRepository).userDetailsService(userDetailsService);

    //H2 CONSOLE
    httpSecurity.authorizeRequests(authorize -> { authorize.antMatchers("/h2-console/**").permitAll(); });
    httpSecurity.headers().frameOptions().sameOrigin();

    //DISABLE CSRF
    httpSecurity.csrf().disable();

    //SECURE EVERYTHING
    httpSecurity.authorizeRequests().anyRequest().authenticated();

    //DEfAULT LOGIN FORM
    httpSecurity.formLogin();

  }

}
```

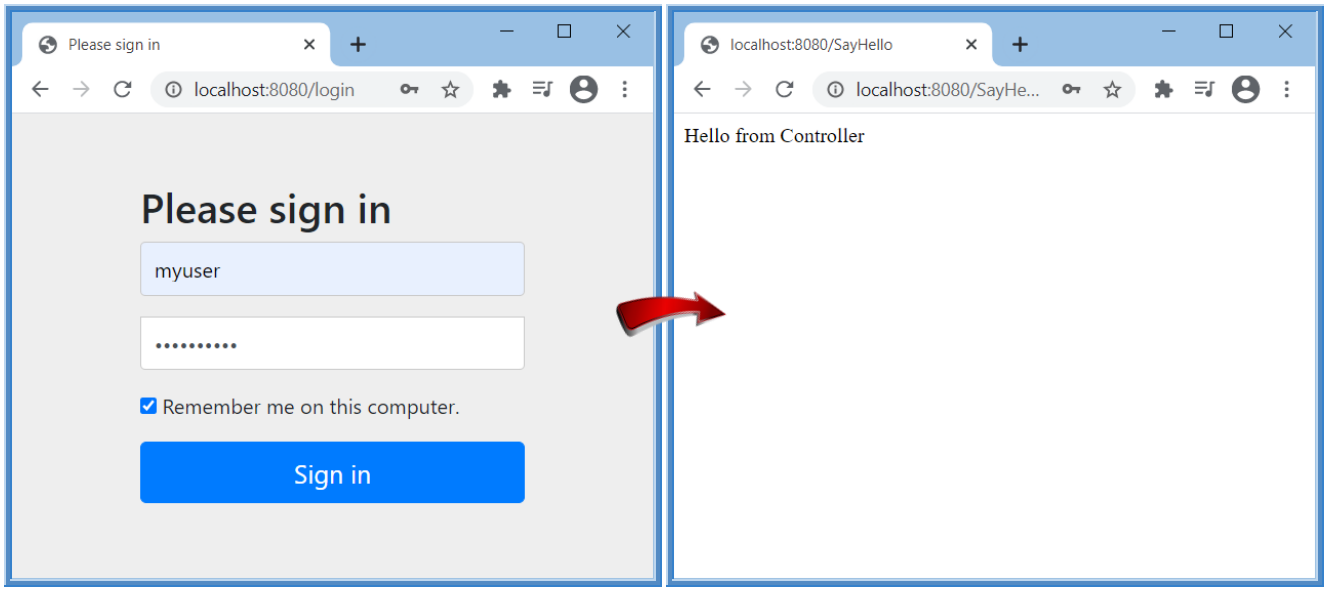
MyController.java

```
package com.ivoronline.springboot_security_rememberme_db.controllers;

import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.ResponseBody;

@Controller
public class MyController {

  @ResponseBody
  @RequestMapping("/SayHello")
  public String sayHello() {
    return "Hello from Controller";
  }
}
```

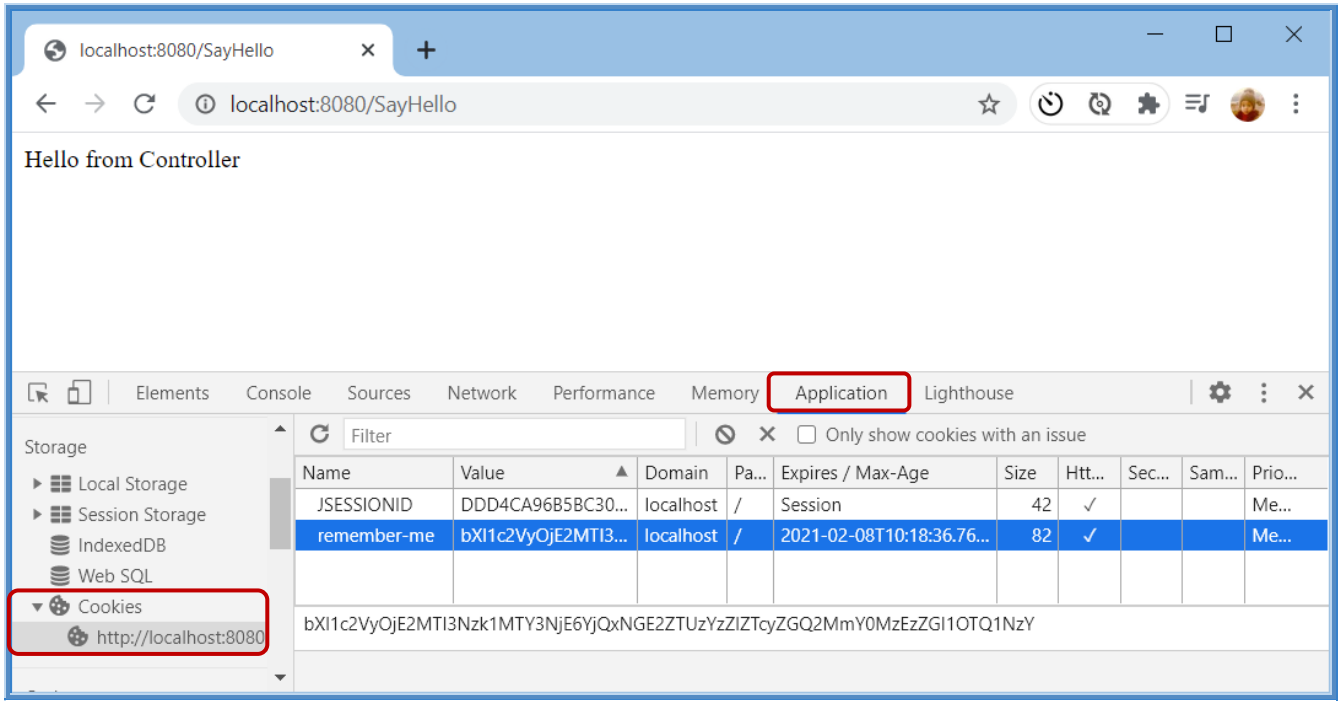
Results

- http://localhost:8080/SayHello
- You get redirected to http://localhost:8080/MyLogin
  - Username: myuser
  - Password: mypassword
  - **Remember Me: CHECK**
  - Sign in
- You get redirected back to http://localhost:8080/SayHello
- Customize and control Google Chrome
  - More Tools
  - Developer Tools
  - Application
  - Cookies
  - http://localhost:8080

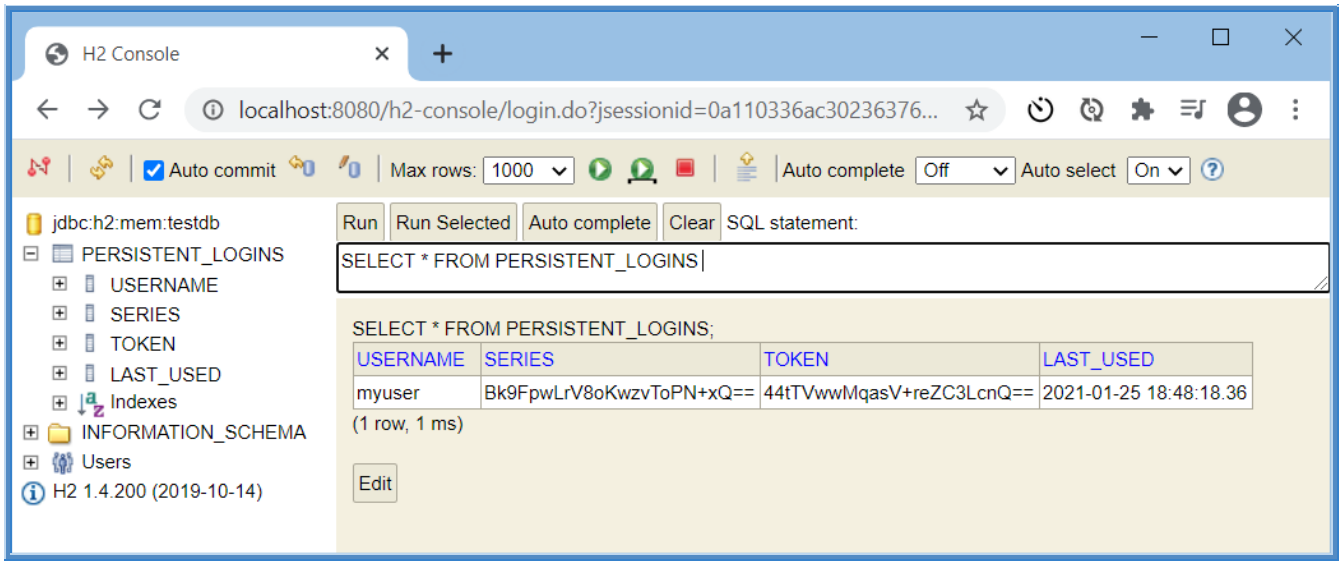
http://localhost:8080/login *http://localhost:8080/SayHello*



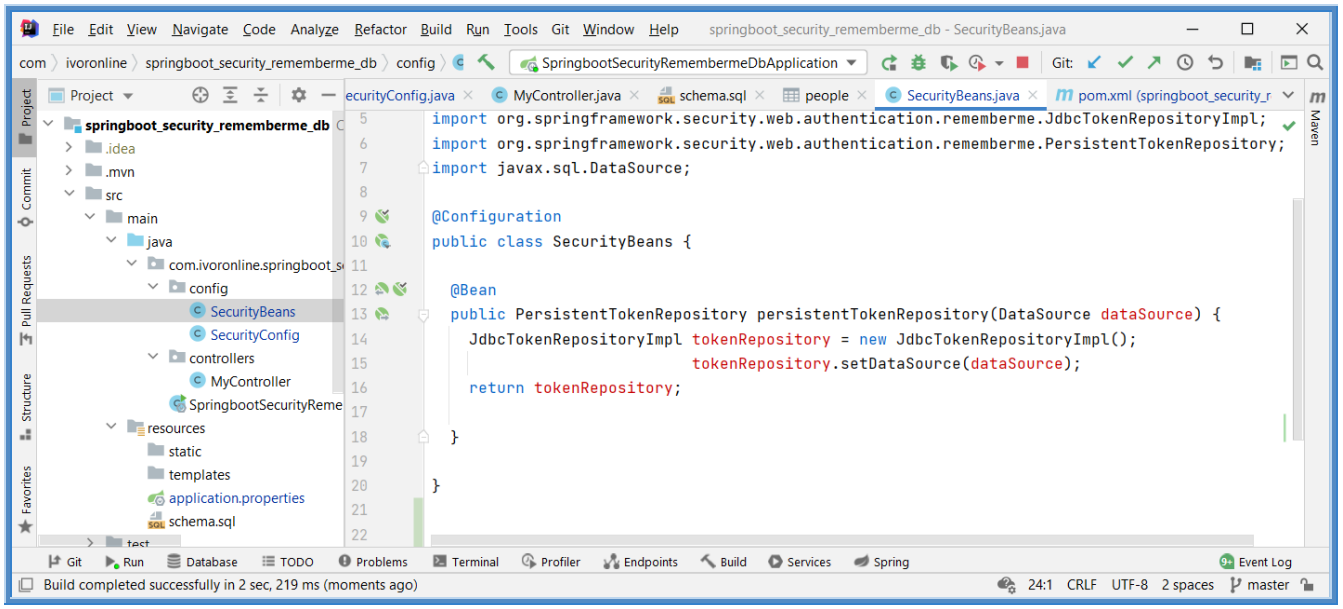
remember-me Cookie



Persistent Token in DB Table: persistent_logins



Application Structure



pom.xml

```
<dependencies>

  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</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-data-jpa</artifactId>
  </dependency>

  <dependency>
    <groupId>com.h2database</groupId>
    <artifactId>h2</artifactId>
    <scope>runtime</scope>
  </dependency>

  <dependency>
    <groupId>org.projectlombok</groupId>
    <artifactId>lombok</artifactId>
    <optional>true</optional>
  </dependency>

</dependencies>
```

4 Appendix

Info

- Following tutorials show how to use additional technologies that can be helpful while developing Spring Boot Application.

4.1 IntelliJ

Info

- Following tutorials show how to use IntelliJ to create Spring Boot Project since all tutorials are done with IntelliJ.
- But you can use any other IDE of your choice.

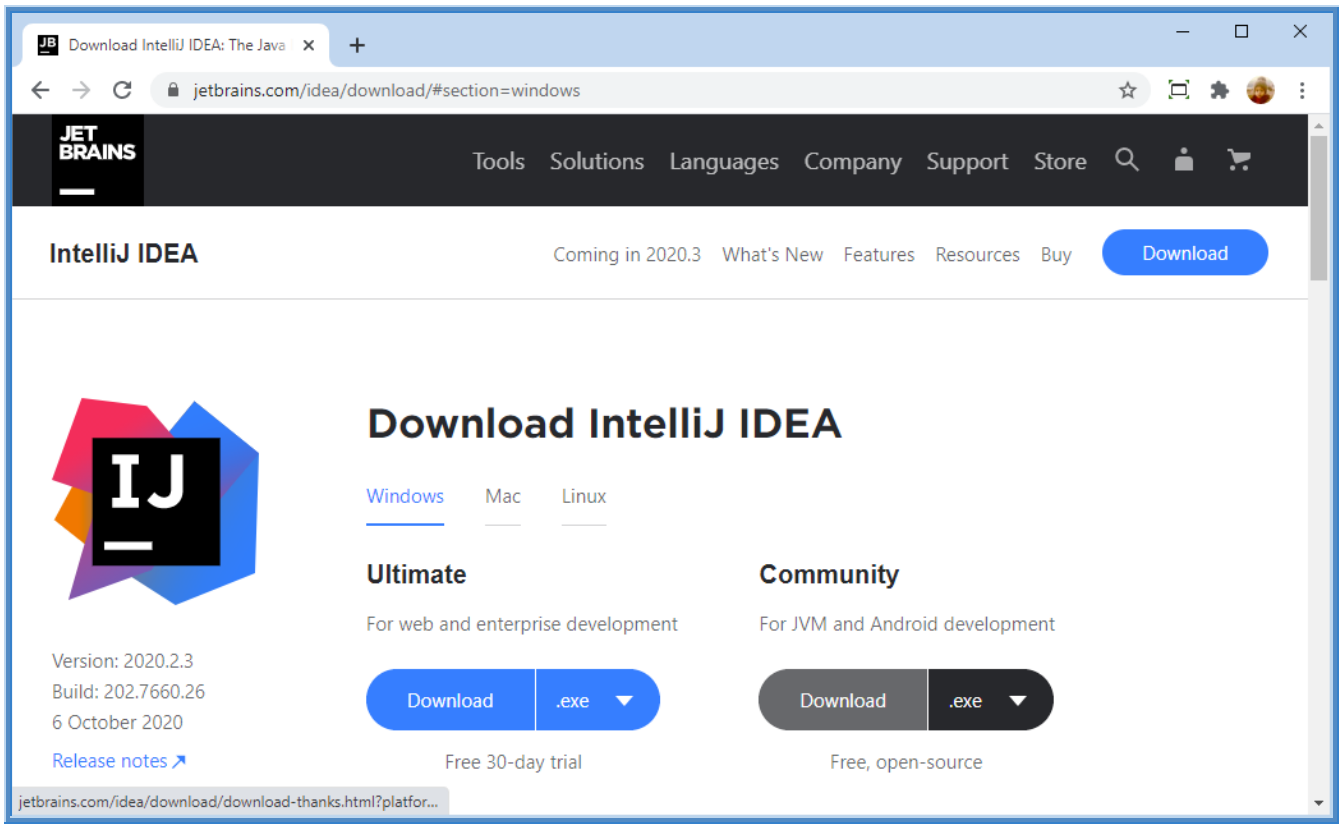
4.1.1 Install

Info

- This tutorial shows how to install IntelliJ IDE Integrated Development Environment (IDE)
  - Download IntelliJ IDEA (free Community or paid Ultimate Edition with free 30 day trial)
  - Install IntelliJ IDEA
  - Download Java JDK (if JDK is not already installed on the PC)
  - Customize (select dark or light interface)

Download IntelliJ IDEA

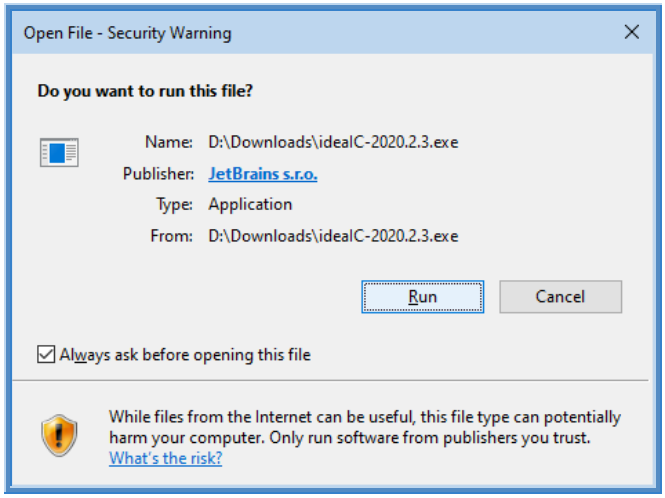
- https://www.jetbrains.com/idea
- Download
- Community
- Download
- Save as: D:\Downloads\idealC-2020.2.3.exe



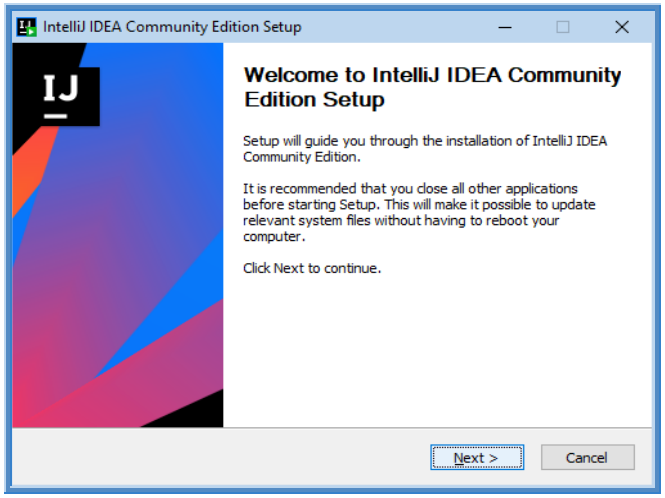
Install IntelliJ IDEA

- Execute idealC-2020.2.3.exe
- Run
- 3*Next

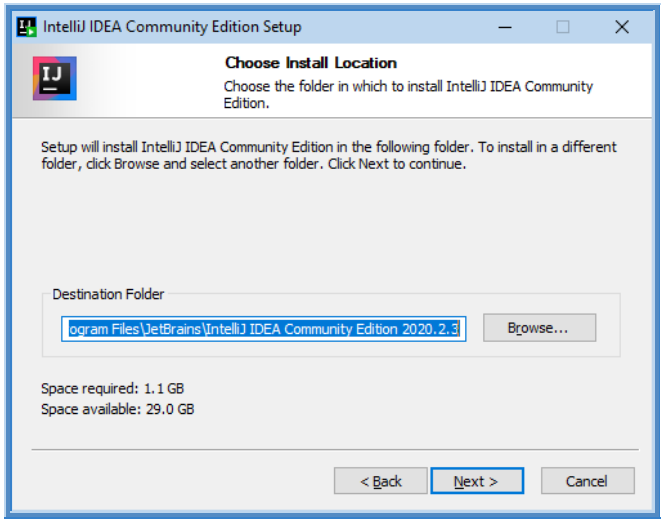
Run



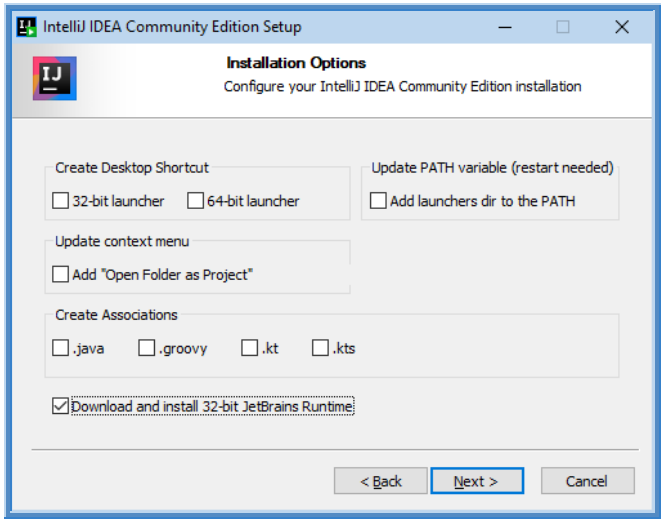
Next



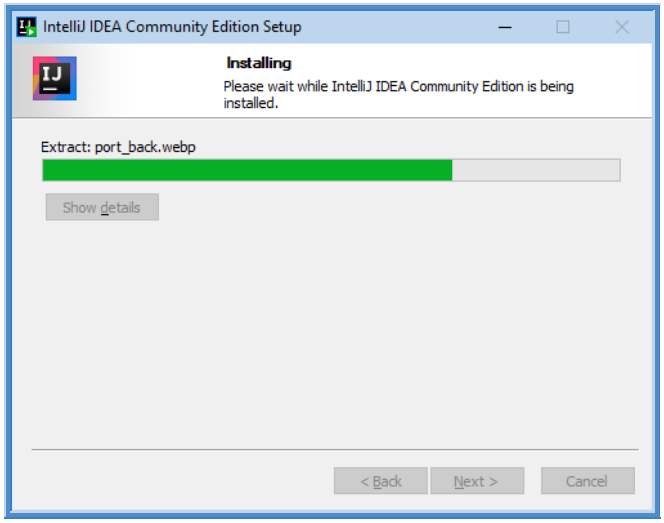
Next



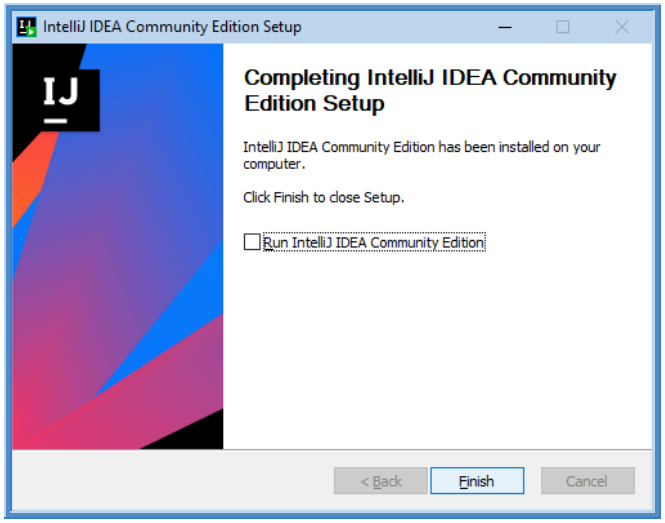
Next



Progress



Finish

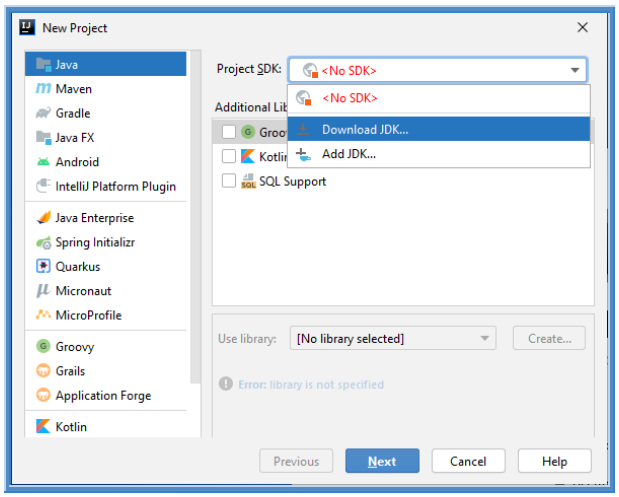


Download Java JDK

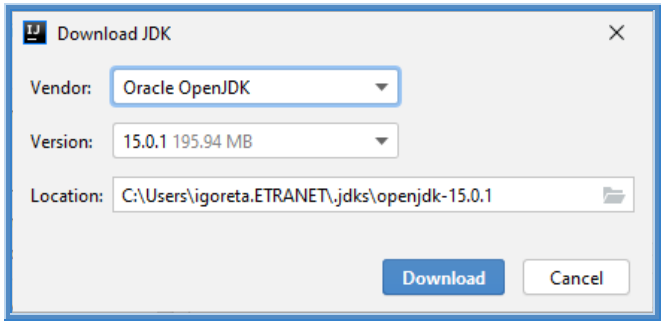
- Start IntelliJ IDEA
- New Project
- Java
- Download JDK

New Project

Download JDK



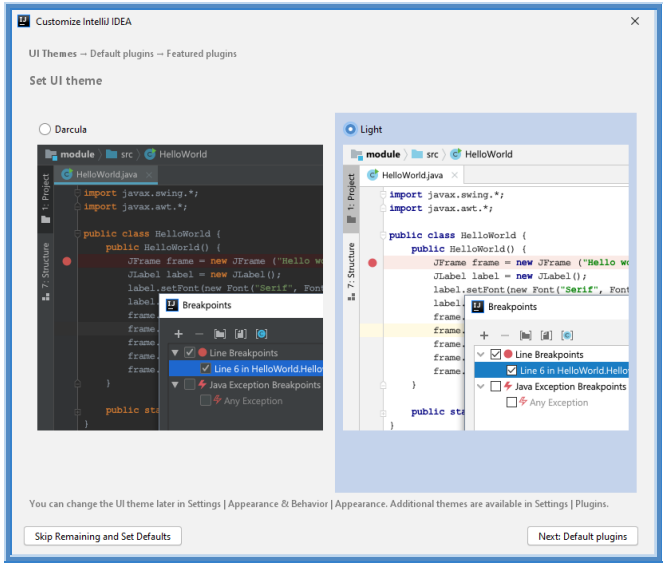
Download



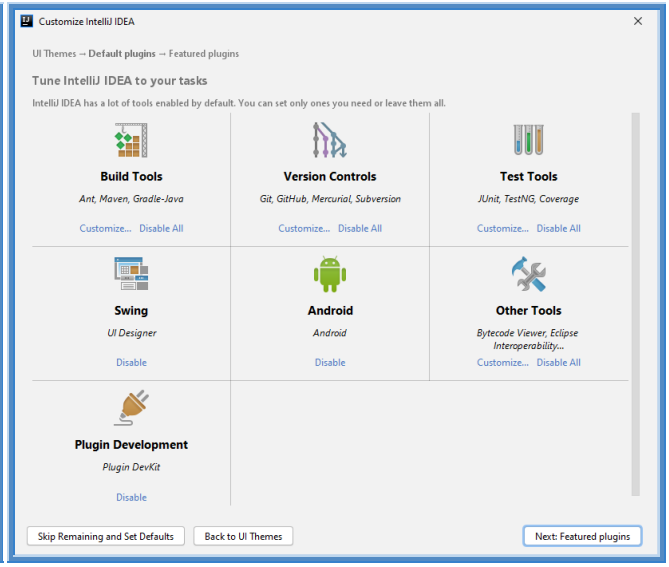
Customize

- Start IntelliJ IDEA
- Light
- 2*Next
- Start

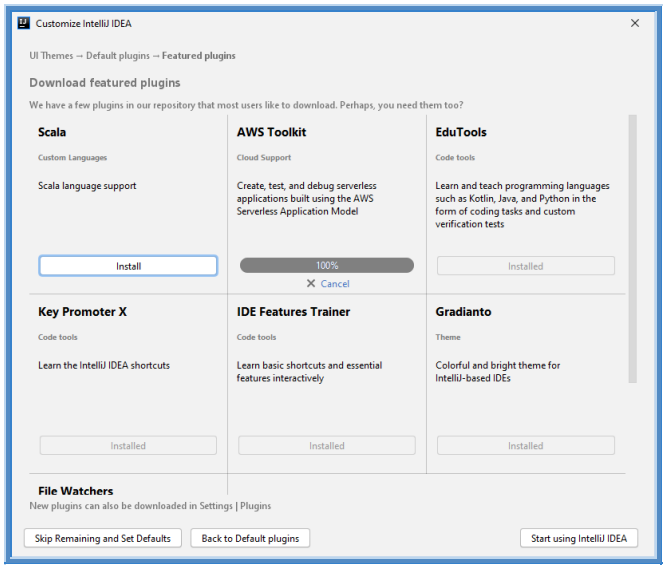
Light



Next



Start



4.1.2 Create Project

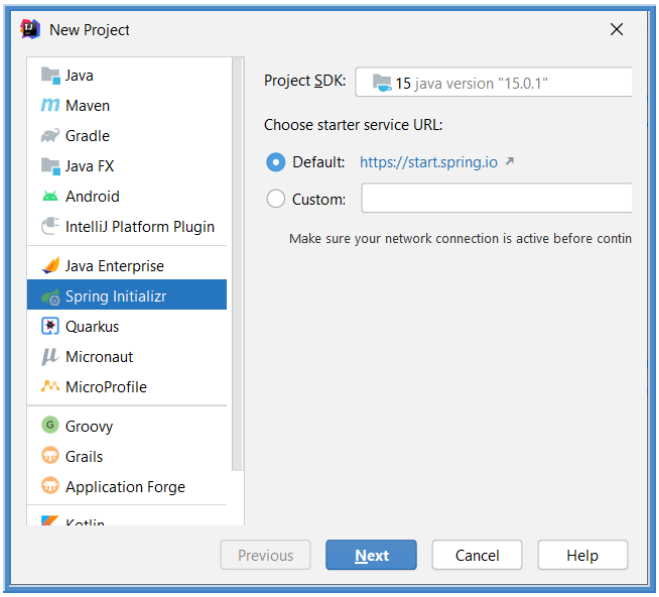
Info

- This tutorial shows how to Create Spring Boot Project using IntelliJ paid **Ultimate** Edition (free 30 day trial).
- In the background IntelliJ uses https://start.spring.io/ Web Application.
- That Web Application is also used in Create Spring Project - Using start.spring.io.

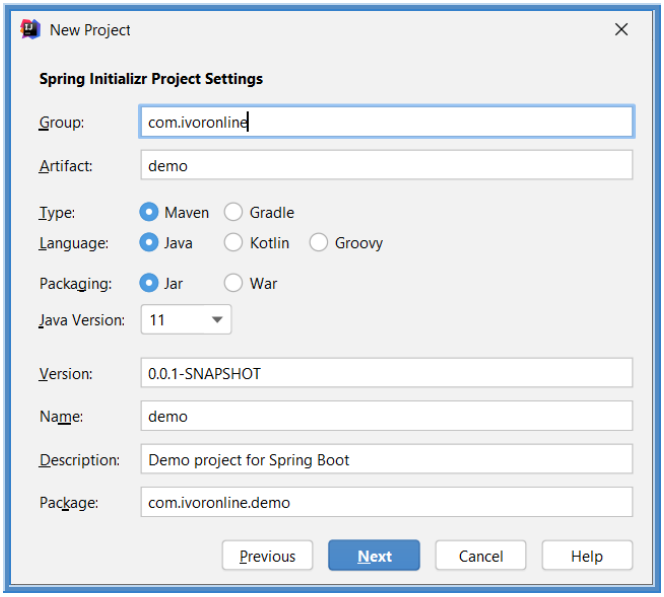
Create Spring Boot Project

- Start IntelliJ
- File - New - Project
- Spring Initializr - Next
- Project Settings
  - Group: com.ivoronline.springboot
  - Artifact: **demo** (Project Name)
  - Type: Maven
  - Language: Java
  - Packaging: Jar
  - Java Version: 11
  - Next
- Dependencies
  - Web: Spring Web
  - SQL: Spring Data JPA
  - SQL: H2 Database
  - Next
- Project name: demo
- Project location: C:\Projects\demo
- Finish
- (wait for it to resolve Maven dependencies - download JARs with Java Classes)

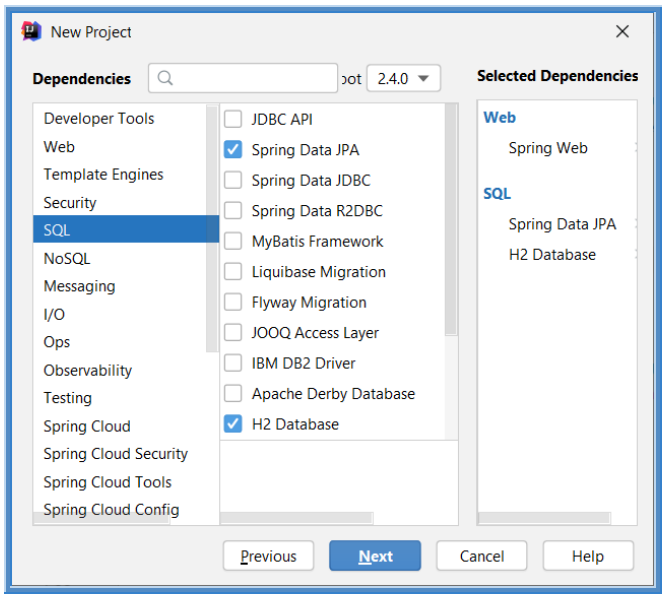
File - New - Project - Spring Initializr



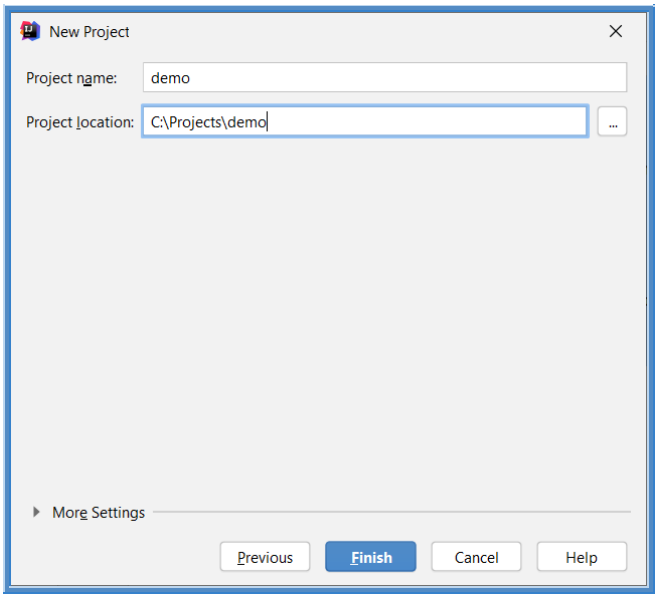
Project Settings



Dependencies



Project name



4.1.3 Run Application

Info

- This tutorial shows how to run Spring Boot Application.
- We will display message in the Console just to make sure everything is working properly.

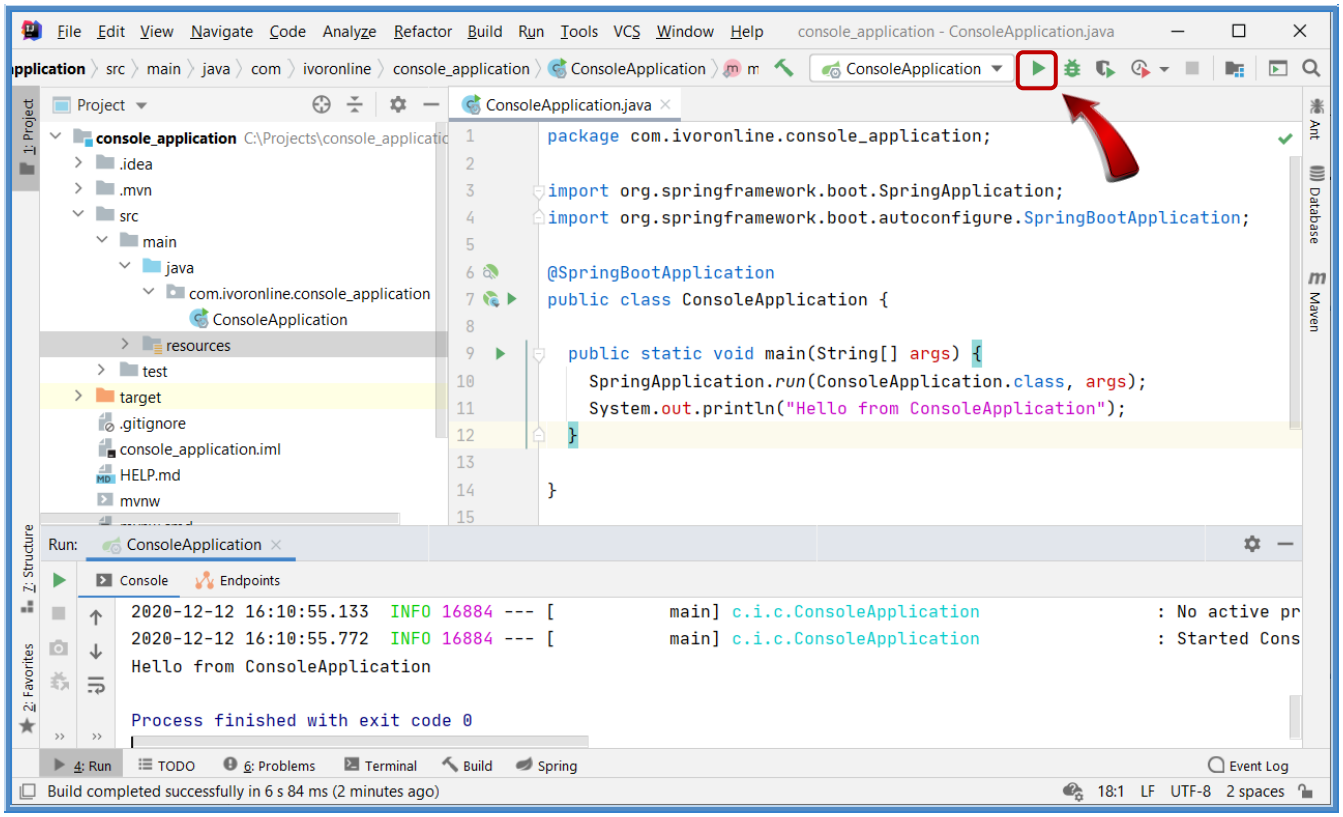
Run Application

- Create Spring Boot Project (no additional dependencies needed)
- Edit Java Class: TestSpringBootApplication.java (insert highlighted line)
- Run Application

ConsoleApplication.java

```
package com.ivoronline.console_application;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class ConsoleApplication {

  public static void main(String[] args) {
    SpringApplication.run(ConsoleApplication.class, args);
    System.out.println("Hello from ConsoleApplication");
  }

}
```

Application



Console

```
Hello from ConsoleApplication
```

5 Summary

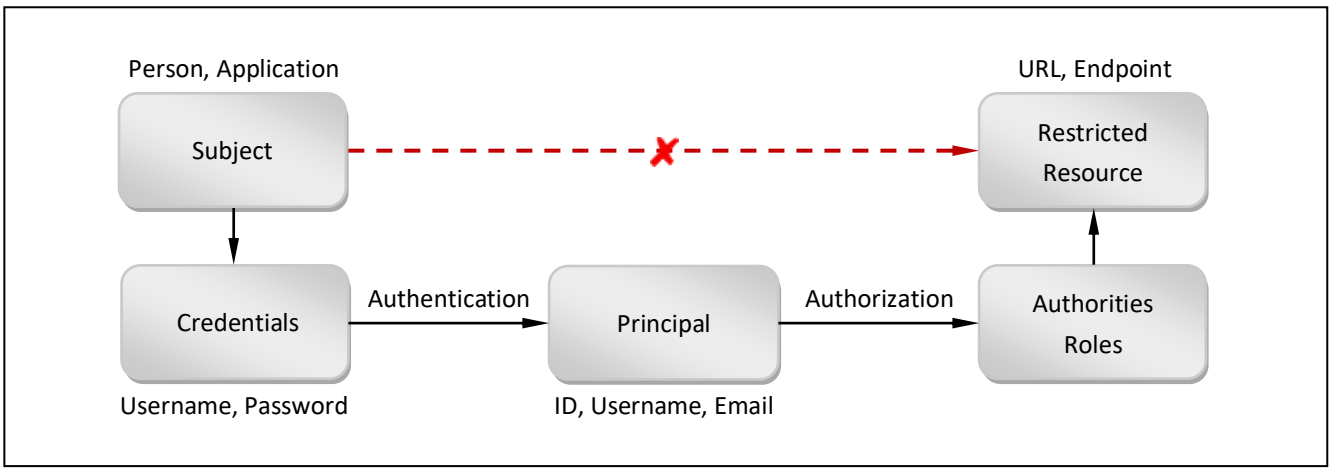
Info

- Security is about preventing Subject to have direct access to Restricted Resource.
- Instead Subject has to
  - **Authenticate** himself (by providing Credentials: "Who are you?")
  - receive **Authorities** (which specify which Restricted Resources it can access: "What are you allowed to do?")

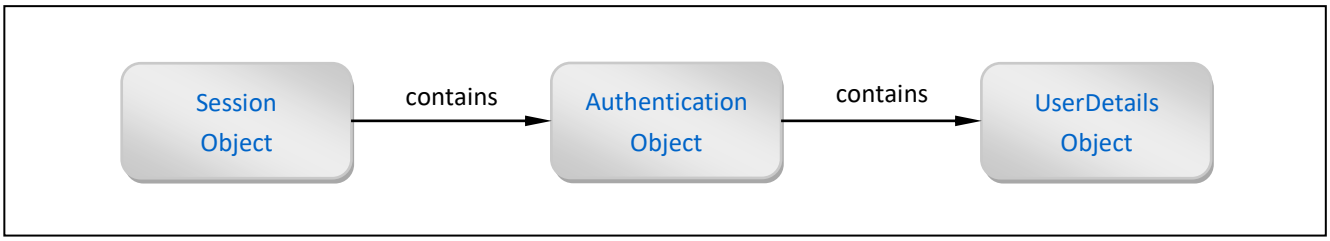
Main Terms

TERM	DESCRIPTION
Subject	Something that wants to access restricted Resource: Person, another Application
Credentials	Used to Authenticate Subject "Who are you?" (Username/Password, Token)
Authentication	Process of uniquely identifying Subject by using Credentials (assigns Identity/Principal to Subject)
Identity/Principal	Something that uniquely identifies Subject: ID, Username, Email, Phone (Result of Authentication)
Authorization	Defines which restricted Resource are accessible to Principal: "What are you allowed to do?"
Authorities/Roles	Control access to Restricted Resources by assigning them to Principals and Restricted Resources.

Main Terms



Main Authentication Objects



5.1 Define Users

application.properties

```
# SECURITY
spring.security.user.name     = myuser
spring.security.user.password = mypassword
spring.security.user.profile  = USER

# PROFILE AUTHORITIES - CRUD
profile.user  = book.read, book.update
profile.admin = book.create, book.read, book.update, book.delete
```

MyUserDetailsService.java

```
@Service
public class MyUserDetailsService implements UserDetailsService {

  //LOAD PROPERTIES (from application.properties file)
  @Value("${spring.security.user.profile}") private String userProfile;
  @Value("${profile.user}")                 private String profileUser;
  @Value("${profile.admin}")                private String profileAdmin;

  @Override
  public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {

    //GET AUTHORITIES FOR GIVEN USER PROFILE
    String userAuthorities = "";
    if(userProfile.equals("USER") ) { userAuthorities = profileUser;  }
    if(userProfile.equals("ADMIN")) { userAuthorities = profileAdmin; }

    //GET AUTHORITIES FROM STRING PROPERTY
    String[]     authoritiesArray = userAuthorities.split(", ");
    List<String> authoritiesList  = Arrays.asList(authoritiesArray);

    //CREATE AUTHORITIES (FOR USER OBJECT)
    List<GrantedAuthority> authorities = new ArrayList<GrantedAuthority>();
    for (String authority : authoritiesList) {
      authorities.add(new SimpleGrantedAuthority(authority.trim()));
    }

    //CREATE USER
    User user = new User("myuser", "mypassword", authorities);

    //RETURN USER
    return user;

  }

}
```

5.1.1 WebSecurityConfig - configure()

WebSecurityConfig - configure()

- In WebSecurityConfig you can Override configure() to specify multiple Users.

WebSecurityConfig.java

```
@Configuration
@EnableWebSecurity
public class WebSecurityConfig extends WebSecurityConfigurerAdapter {

  @Override
  protected void configure(AuthenticationManagerBuilder auth) throws Exception {
    auth.inMemoryAuthentication().withUser("myadmin").password("{noop}myadminpassword").roles("ADMIN");
    auth.inMemoryAuthentication().withUser("myuser" ).password("{noop}myuserpassword" ).roles("USER" );
  }

}
```

5.2 Add Authorities to Endpoints

Authorities

- This summary shows different ways if assigning Authorities/Roles to Endpoints.
  Since Authorities/Roles are also assigned to Users this defines which Users have access to which Endpoints.
- **Authorities** control which Endpoints User can call (book.creat, book.read, ROLE_ADMIN, ROLE_USER)
  - **UserDetails** contains list of Authorities that are assigned to that User
  - **Endpoint** contains list of Authorities that User must have to call it
- **Roles** are just Authorities with prefix **ROLE_**. (ROLE_ADMIN, ROLE_USER)
- If Roles & Authorities are added to Endpoints that doesn't mean they will be used.
  This is because in WebSecurityConfig you can define how to control access to Endpoints.
  For **denyAll()**, **permitAll()** and **authenticated()** Spring only looks if User is Authenticated. (ignoring Roles and Authorities)
- If Roles & Authorities are not added to Endpoints then by default User only needs to be Authenticated. (no Authorities)

5.2.1 Annotations

Info

- You can use assign Authorities to Endpoints by using
  - @Secured Annotation to assign Roles to single Endpoint
  - @PreAuthorize Annotation to assign Roles & Authorities to multiple Endpoints

Endpoint Annotations

NAME	DESCRIPTION
@Secured ("ROLE_ADMIN")	Authority ROLE_ADMIN is required to call Endpoint
@Secured ({"ROLE_ADMIN", "ROLE_USER"})	Authority ROLE_ADMIN or ROLE_USER is required to call Endpoint
@PreAuthorize ("hasRole('ADMIN')")	Authority ROLE_ADMIN is required to call Endpoint
@PreAuthorize ("hasAnyRole('ADMIN', 'USER')")	Authority ROLE_ADMIN or ROLE_USER is required to call Endpoint
@PreAuthorize ("hasAuthority('ROLE_ADMIN')")	Authority ROLE_ADMIN is required to call Endpoint

WebSecurityConfig.java *(enable Annotations)*

```
@EnableGlobalMethodSecurity(securedEnabled = true, prePostEnabled = true)  //Enables @Secured & @PreAuthorize
```

```
public class WebSecurityConfig extends WebSecurityConfigurerAdapter { ... }
```

MyController.java *(assign Roles & Authorities to single Endpoint)*

```
@RestController
public class MyController {

  @Secured("ROLE_ADMIN")
  @PreAuthorize("hasRole('ADMIN')")
  @RequestMapping("/Hello")
  public String hello() {
    return "Hello from Controller";
  }

}
```

5.2.2 Annotations - Custom Method

@PreAuthorize - Custom Methods

- **@PreAuthorize** Annotation can also have an additional Custom Method to control access to Endpoint.

MyController.java

```
@RestController
public class MyController {

  @PreAuthorize("hasRole('USER') AND @customAuthorizationService.authorize(authentication)")
  @RequestMapping("/Hello")
  public String hello() {
    return "Hello from Controller";
  }

}
```

CustomAuthorizationService.java

```
@Component
public class CustomAuthorizationService {

  //================================================================================
  // AUTHORIZE
  //================================================================================
  public boolean authorize(Authentication authentication) {

    //GET USERNAME
    UserDetails userDetails = (UserDetails) authentication.getPrincipal();
    String      userName    = userDetails.getUsername();

    //CHECK USERNAME
    return userName.equals("myuser");

  }

}
```

5.2.3 antMatchers()

Info

- You can use assign Roles & Authorities to multiple Endpoints by using inside WebSecurityConfig.cofigure() Method.
- To **select** Endpoints you can use either
  - **anyRequest()** to select **all** Endpoints (must be used after antMatchers())
  - **antMatchers()** to select only Endpoints with **matching URL** (last one is taken into account)
- You can control access to selected Endpoints depending if User is **Authenticated** (ignoring Roles and Authorities)
  - **denyAll()** No access to selected Endpoints (even after login/Authentication)
  - **permitAll()** Anonymous access to selected Endpoints (no needed for login/Authentication)
  - **authenticated()** Authenticated access to selected Endpoints (User needs to login/Authenticate)
- Or you can **add Roles & Authorities** to selected Endpoints by using
  - **hasRole("ADMIN")** Add ADMIN_ROLE to selected Endpoints
  - **hasAnyRole("ADMIN", "USER")** Add ADMIN_ROLE & USER_ROLE to selected Endpoints
- You also need to specify **how to Authenticate** Users
  - **httpSecurity.formLogin()** Authenticate using default Login Form
- If multiple Matchers are used for the same Endpoint only the last one will be taken into account.
  Roles that they assign are not cumulative - only the last ones will be used.

WebSecurityConfig.java *(add Authorities to Endpoints)*

```
@Configuration
public class WebSecurityConfig extends WebSecurityConfigurerAdapter {

  @Override
  protected void configure(HttpSecurity httpSecurity) throws Exception {

    //ADD ROLES AND AUTHORITIES TO ENDPOINTS                              //Alternative is Annotations
    httpSecurity.authorizeRequests()
      .anyRequest()                          .hasRole   ("ADMIN")          //Add Role  to all Endpoints
      .antMatchers("/endPoint1")             .hasRole   ("ADMIN")          //Add Role  to selected Endpoint
      .antMatchers("/endPoint2", "/endPoint3").hasRole   ("ADMIN")          //Add Role  to selected Endpoints
      .antMatchers("/cars/*"   , "/planes/*" ).hasRole   ("USER");          //Add Role  to selected Endpoints
      .antMatchers("/endPoint4")             .hasAnyRole("ADMIN", "USER"); //Add Roles to selected Endpoint

    //SPECIFY ACCESS BASED ONLY ON AUTHENTICATION (ignore Roles & Authorities)
    httpSecurity.authorizeRequests()
      .antMatchers("/endPoint1").denyAll()             //No            access (even after Login)
      .antMatchers("/endPoint2").permitAll()           //Anonymous     access (no Login needed)
      .antMatchers("/endPoint3").authenticated();      //Authenticated access (ignores Roles & Authorities)
      //If not specified for Endpoint                  //Authenticated access (checks  Roles & Authorities)

    //AUTHENTICATE WITH DEFAULT LOGIN FORM
    httpSecurity.formLogin();                          //Without this there is no Login Form to Authenticate

  }

}
```

URL Patterns

PATTERN	EXAMPLES
"/endPoint1"	Specific URL or Pattern
"/endPoint1", "/endPoint2"	List of specific URLs or Patterns
"/*"	All URLs on this level: /hello, /endPoint123, /sublevel/endPoint4
"/end*"	All URLs on this level that start with end: /hello, /endPoint123, /sublevel/endPoint4
"/end/*"	All URLs on sub level that start with end: /hello, /end/Point1, /sublevel/endPoint4
"/**"	All URLs on this or lower levels: /hello, /endPoint123, /sublevel/endPoint4
"/end**"	All URLs on this or lower levels that start with end: /hello, /endPoint123, /end/endPoint4
"/end/**"	All URLs on sub level that start with end: /hello, /end/Point1, /end/start/endPoint4
HttpMethod.GET, "/end*"	All Endpoints that accept GET and start with /end (@RequestMapping, @GetMapping)

5.3 Authentication Classes & Objects

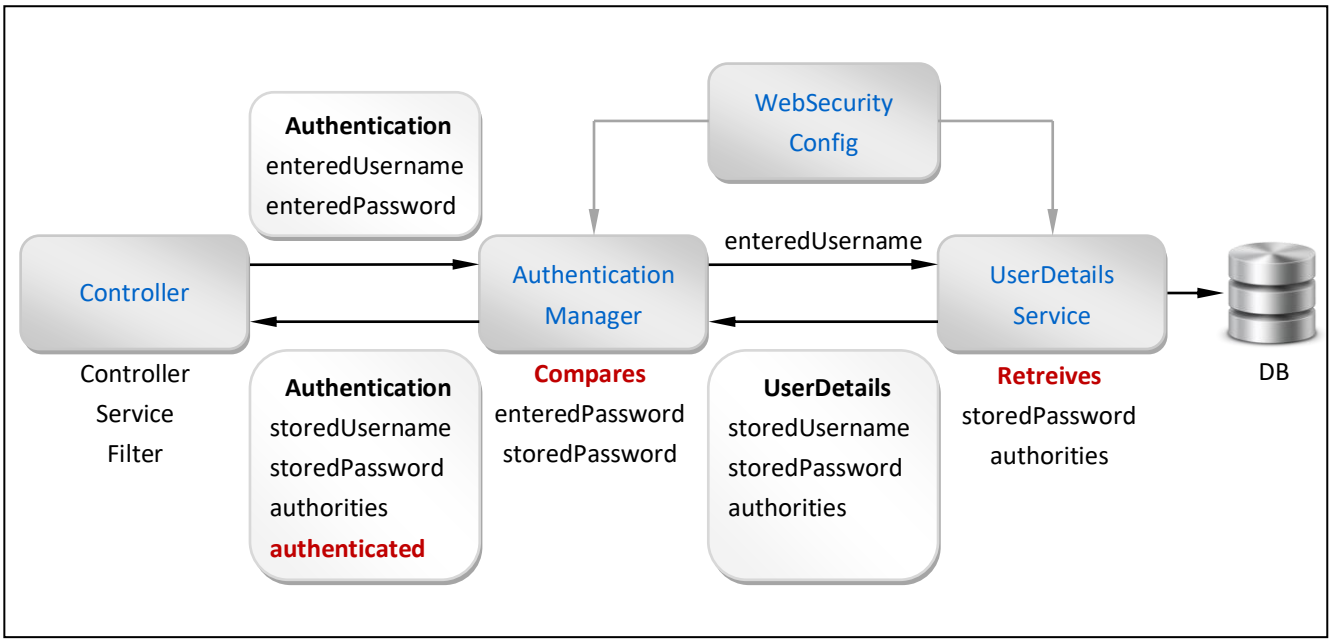
Info

- Classes (that implement Methods needed for Authentication)
  - **UserDetailsService** uses **enteredUsername** to return **UserDetails** with **storedPassword** & **authorities**
  - **AuthenticationManager** uses **enteredUsername** & **storedPassword** from **UserDetails** to return **Authentication**
  - **WebSecurityConfig** is used to define Users, Password Encoder, add Authorities to Endpoints, ...
- Objects (that store data needed for Authentication)
  - **UserDetails** is **DTO** that transfers Stored Username, Password & Authorities from DB to **Authentication**
  - **Authentication** is used by Spring to control access to Endpoints using **Authorities** and **authenticated = true**

Authentication Process

- Code that initiates Authentication (Controller in below Schema)
  - reads **enteredUsername** and **enteredPassword** (from HTTP Request Parameters or Headers)
  - creates **Authentication** Object with **enteredUsername** and **enteredPassword**
  - calls **AuthenticationManager authenticate()** Method with **Authentication** Object as Input Parameter
  - stores returned **Authentication** Object in Context (so that Authorities can be used to access Restricted Resources)
- **AuthenticationManager**
  - receives **Authentication** Object as Input Parameter (with enteredUsername & enteredPassword)
  - calls **UserDetailsService** with **enteredUsername**
  - from **UserDetailsService** it gets **UserDetails** Object with **storedPassword** and **authorities**
  - compares **enteredPassword** with **storedPassword**
  - If Passwords match User is considered Authenticated and **AuthenticationManager** returns **Authentication** Object with
    - **storedUsername**
    - **storedPassword**
    - **authorities** (taken from UserDetails Object)
    - **authenticated = true**

Authentication Process



5.3.1 MyUserDetailsService

Info

- **UserDetailsService** is used to get **storedPassword** for given **enteredUsername**
  - **UserDetailsService** only gets **enteredUsername** as input parameter
  - Then it looks for the User with that username in the Database
  - If such **storedUsername** exists it gets related **storedPassword** and **authorities**
  - Then it stores **storedUsername**, **storedPassword** and **authorities** in returned **UserDetails** Object
- **UserDetailsService** doesn't know about **enteredPassword** and therefore it can't Authenticate the User.
  Instead returned **UserDetails** Object will be used by **AuthenticationManager** to compare **enteredPassword** with the **storedPassword** in returned **UserDetails** Object.
- As part of Authentication Process AuthenticationManager uses MyUserDetailsService by calling its
  - loadUserByUsername() Method with **Entered** Username
  - which returns UserDetails Object with **Stored** Username, Password, Authorities
  - AuthenticationManager compares Entered & Stored Password to Authenticate User

MyUserDetailsService.java *(get Users/Accounts from DB)*

```
@Service
public class MyUserDetailsService implements UserDetailsService {

  @Autowired AccountRepository accountRepository;

  //=========================================================================
  // LOAD USER BY USERNAME
  //=========================================================================
  @Override
  public UserDetails loadUserByUsername(String enteredUsername) throws UsernameNotFoundException {

    //-------------------------------------------------------------------------------------------
    //MOCK DB: <username, {"password", "ROLE"}>
    HashMap<String, String[]> accounts = new HashMap<>();
                              accounts.put("myuser" , new String[]{"myuserpassword" , "ROLE_USER" });
                              accounts.put("myadmin", new String[]{"myadminpassword", "ROLE_ADMIN"});
    //-------------------------------------------------------------------------------------------

    //GET USER/ACCOUNT (From DB)
    String[] account = accounts.get(enteredUsername);
    Account  account = accountRepository.findByUsername(enteredUsername);

    //CHECK IF USER/ACCOUNT EXISTS
    if (account == null) { throw new UsernameNotFoundException(enteredUsername); }     //Bad credentials

    //GET PASSWORD
    String storedPassword = account.password;                                          //account[0];

    //GET AUTHORITIES
    List<GrantedAuthority> authorities = new ArrayList<>();
                           authorities.add(new SimpleGrantedAuthority(account.role));  //account[1]

    //CREATE USER DETAILS OBJECT
    UserDetails userDetails = new User(enteredUsername, storedPassword, authorities);

    //RETURN USER DETAILS OBJECT
    return userDetails ;

  }

}
```