



SPRING BOOT

SECURITY

Version: 1.0

Date: 03.2021

www.ivoronline.com

1 MAIN TERMS	6
1.1 Theory	7
1.1.1 Authentication.....	8
1.1.2 Authentication - Identity/Principal.....	9
1.1.3 Authentication - Credentials vs Principal.....	10
1.1.4 Authentication - User Object.....	11
1.1.5 Authentication - Session	12
1.1.6 Authorization.....	14
1.1.7 Authorization - Authorities vs Roles	15
1.1.8 Security - Different Implementations	16
1.1.9 Security - WebSecurityConfig.java.....	17
1.1.10 Security - MyUserDetailsService.java	18
1.1.11 Security - UserDetailsService vs AuthenticationManager	20
1.2 Define Users	23
1.2.1 Default User.....	24
1.2.2 application.properties.....	27
1.2.3 API - userDetailsService()	30
1.2.4 API - configure()	34
1.2.5 DB.....	38
1.3 Authentication	47
1.3.1 Automatic - Login Form - Default	48
1.3.2 Automatic - Login Form - Custom.....	51
1.3.3 Automatic - Authorization Header - Postman - Add.....	56
1.3.4 Automatic - Authorization Header - Postman - Generate	58
1.3.5 Validate Credentials - Request - GET	60
1.3.6 Validate Credentials - Request - POST	64
1.3.7 Validate Credentials - Request - POST - JSON	69
1.3.8 Manually Create User Object	74
1.3.9 Events - Log to Console	79
1.4 Authorization	84
1.4.1 Security Expressions - API - Roles & Authorities	86
1.4.2 Security Expressions - @Secured - Roles	89
1.4.3 Security Expressions - @PreAuthorize - Roles & Authorities.....	92
1.4.4 Security Expressions - @PreAuthorize - Custom Methods	95
1.4.5 Security Expressions - @PreAuthorize - Custom Methods - Books.....	100
1.4.6 URL Patterns - Ant Matchers	108
1.4.7 Roles.....	113
1.4.8 Authorities - application.properties	116
1.4.9 Authorities - DB	121
1.5 Password Encoders.....	134
1.5.1 No Operation	135
1.5.2 LDAP	140
1.5.3 Sha256.....	145
1.5.4 BCrypt.....	150
1.6 Manual Authentication	155
1.6.1 Theory - Classes & Interfaces	157
1.6.2 Single Time (Session Based) - Request Parameters.....	160

1.6.3	Every Time (Filter Based) - Request Parameters	166
1.6.4	Every Time (Filter Based) - Request Headers	171
1.7	CSRF (Cross-Site Request Forgery).....	177
1.7.1	Theory - Normal User Interaction.....	178
1.7.2	CSRF Attack - Theory	180
1.7.3	CSRF Attack - Application	182
1.7.4	CSRF Token - Theory	183
1.7.5	CSRF Token - Application	184
1.8	Remember Me	188
1.8.1	Login Form - Default	189
1.8.2	Login Form - Custom	193
1.8.3	Login Form - Default - DB - PostgreSQL	198
1.8.4	Login Form - Default - DB - H2.....	203
1.9	CORS (Cross Origin Resource Sharing).....	209
1.9.1	Application - Source	210
1.9.2	Application - Destination.....	212
1.9.3	CORS - Disabled	214
1.9.4	CORS - Enabled - Annotations	216
1.9.5	CORS - Enabled - API	218
1.10	2FA (2 Factor Authentication)	220
1.10.1	Step 1: Copy Project.....	221
1.10.2	Step 2: Register.....	224
1.10.3	Step 3: Enter Code	231
1.10.4	Step 4: Restrict Access	235
1.11	JWT (JSON Web Token)	239
1.11.1	Step 1 - Get Token	248
1.11.2	Step 2 - Send Token - As Request Parameter - Get Claims	252
1.11.3	Step 3 - Send Token - In Authorization Header - Get Claims	256
1.11.4	Step 4 - Send Token - In Authorization Header - Get Username	259
1.11.5	Step 5 - Filter	264
1.11.6	Step 6 - Authenticate	269
2	DEMO APPLICATIONS	275
2.1	JWT Authorities from DB.....	276
2.1.1	Step 1 - DB Authorities.....	277
2.1.2	Step 2 - JWT	278
3	APPENDIX	288
3.1	IntelliJ	289
3.1.1	Install.....	290
3.1.2	Create Project	294
3.1.3	Run Application.....	296
3.2	JWT.....	297
3.2.1	Introduction.....	298
3.2.2	Structure.....	299
3.2.3	Usage.....	301
3.2.4	Encode/Decode	302

ABOUT THIS 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 quickly get you started with Spring Boot security like: Authentication, Authorization, Roles, Authorities, Credentials, Login Form, Username, Password, CSRF, CORS, Remember Me, 2FA, JWT.

Standalone Tutorials

The core of this Book are standalone tutorials that explain different functionalities of Spring Boot.

Each tutorial contains minimum amount of code needed to explain specific functionality. Tutorials have minimum amount of encompassing text that explains related theory and different parts of the code. This approach allows students to grasp presented concepts in a very fast and efficient manner. Full code, which can also be downloaded from GitHub, prevents any time being wasted trying to make the code work. Simple examples allow for full understanding of the functionality without any unnecessary distractions.

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 practical applications.

Demo Application

Book contains demo Application that show how to combine some of the security features covered in 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!

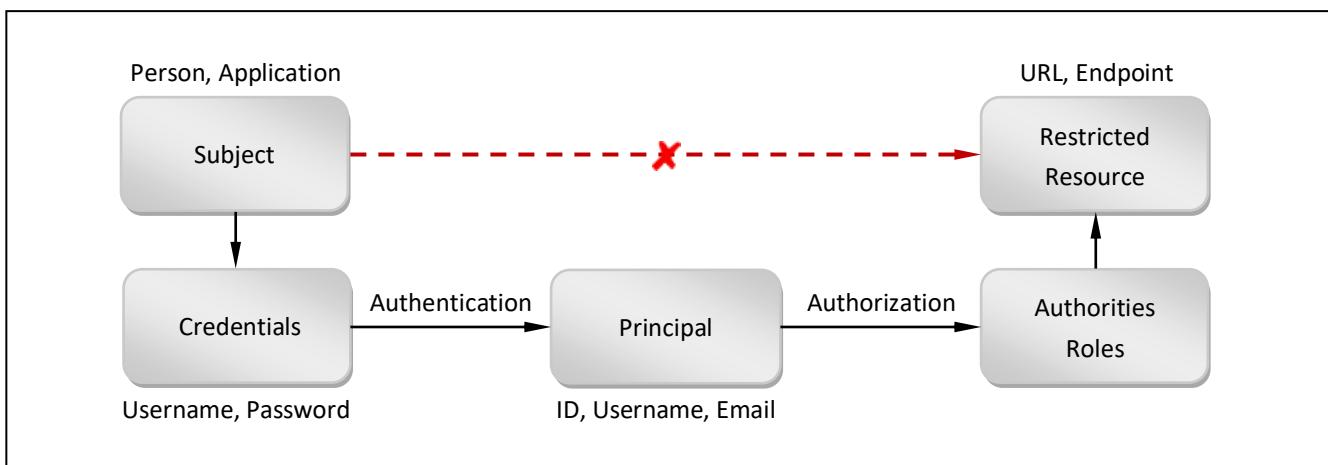


1 Main Terms

Info

- Security is all about preventing Subject to have direct access to Restricted Resource (indicated with red dashed line).
- Instead Subject has to take a longer root
 - by first providing Credentials in order to **Authenticate** himself
 - and then receiving **Authorities** which specify which Restricted Resource it can access
- **Subject** is something that wants to access restricted Resource like
 - **Person, User, Process, Application**
- **Restricted Resource** is something that can be accessed only by specific Subjects and can be
 - **Room, Application, URL, Endpoint**
- **Credentials** are security related items used to Authenticate Subject. They answer question: "Who are you?". They can be
 - **Username, Password, Temporary Code, ID Card, Bank Card, Token**
- **Authentication** is process of uniquely identifying Subject by using Credentials.
 - Authentication answers the question: "Who are you?" => By using Credentials
- **Identity/Principal** is something that uniquely identifies Subject (after it has been Authenticated) like
 - **ID, Username, Email, Phone Number**
- **Authorization** defines which restricted Resource are accessible to Subject/Principal
 - Authorization answers question: "What are you allowed to do?" => By using Authorities & Roles
- **Authorities/Roles** are assigned to both Principals and Restricted Resources to control access to Restricted Resources.

Main Terms



1.1 Theory

Info

- Following tutorials describe theory behind tutorials that will cover Spring Boot Security.

1.1.1 Authentication

Info

- **Authentication** answers question: "Who are you?" => By using Credentials: Username, Password, Temporary Code
- **Authentication** concerns itself with how to
 - define Users (application.properties, Class, Database)
 - enable Users to provide their Credentials (Login form, Authentication Header)
- **Authentication** can be
 - **Database Authentication** if Users are defined in **DB**
 - **In-memory Authentication** if Users are defined in **Application**
 - **Default** User and autogenerated Password user/506e6f00-2b11-4036-96d6-74633e94da2d
 - **Single** User defined in **application authorities** spring.security.user.name / password / roles
 - **Multiple** Users defined in **SecurityConfig Class** .username("myuser").password("mypass").roles("ADMIN")
- After User gets Authenticated, **User Object** is created to hold User data (Username, Password, Authorities).
This User Object is then used to Authorize access to different Endpoints based on User's Authorities (Roles).

1.1.2 Authentication - 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 will create Session in which it stores unique **Identity/Principal** so that the Application knows which Subject is sending 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 Identities/Principals
 - First Jack Carpenter holds ID = 100 in Application's DB
 - Second Jack Carpenter holds ID = 200 in Application's DBSo in this example Identity/Principal is their unique ID from Database.

1.1.3 Authentication - 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.

1.1.4 Authentication - User Object

Info

- When User gets successfully Authenticated Spring creates **User Object** to hold User data which
 - is stored in memory inside User Session
 - is identified with JSESSIONID Cookie that is returned by Server (and stored in Browser as explained in [Session Object](#))
 - can be instantiated either from
 - built-in **User** Class (which also has some additional Properties as defined by Interfaces that it must Implement)
 - some Custom Class (which must Implement Interface, has additional Properties compared to built-in User Class)

Create User Object - from Built-in User Class

- Once the User gets Authenticated (against its Username and Password) instance of built-in User Class is created.
- This Object contains: Username, Password, Authorities and some additional Properties.
- You can reference this User Object inside the Controller if needed.

Create User Object - from Custom Class

- Once the User gets Authenticated (against its Username and Password) instance of Custom User Class can be created.
- This Object contains all the Properties of Built-in User plus any additional Properties that you specify and set like
 - Role
 - UserId
 - CustomerId (group of Users belonging to the same company)
- You can reference this User Object inside the Controller in the same way as Built-in User.
This is because Spring will simply use your Custom User in place of Built-in User.
To declare Custom User your Class just needs to Implement specific Interfaces.

Usage

- User Object can be accessed inside the Controller (to implement Custom Authorization or for other purposes)

1.1.5 Authentication - Session

Info

- **Session Object**
 - is Java Object
 - is stored on the Server
 - is used to track different Users
 - contains User related data

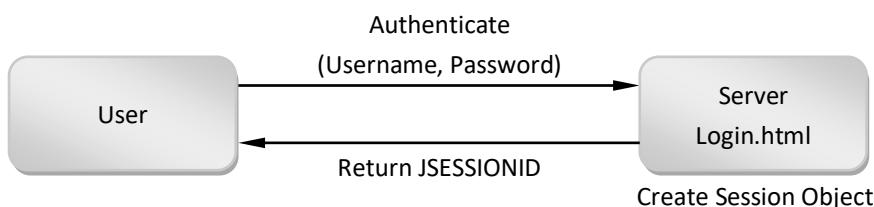
(either in the Memory or in the Database)
(different Session Object is assigned to each User)
(like is User Authenticated, User's Role and Username)
- **JSESSIONID**
 - is Cookie
 - is stored in the Browser
 - is used to track different Users
 - contains ID that uniquely identifies Session Object
 - is received from Server after successful Authentication
 - is sent to Server with every HTTP Request
 - is used by Server to identify User's Session Object

(different JSESSIONID is returned to each User)
(assigned to that User)
(after User provided valid Username and Password)
- **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) in order to allow User access to restricted Resource in subsequent TTP 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**.

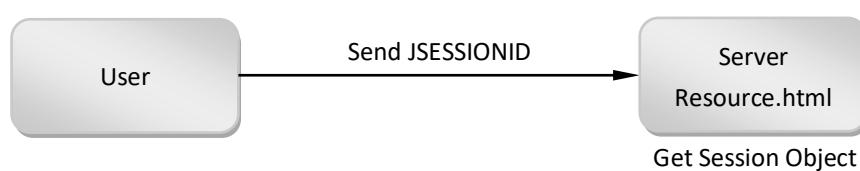
Session Object Example

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

Initial HTTP Request



Subsequent HTTP Requests



Store User Data in additional Browser Cookies

- Alternatively data stored in the Session Object could be **stored in additional Browser's Cookies** and also sent with every Request. But this might have following negative effects
 - This might allow User to temper with the data (for instance changing User Role from **USER** to **ADMIN**) and that way getting access to unauthorized resources. For that reason keeping such User data on the Server is more secure.
 - Since JSESSIONID just contains ID, it is very small and therefore convenient to send with each HTTP Request. Storing all of User data in additional Cookies (as suggested above) and then sending them with every HTTP Request might slow down interaction with Server because more data would need to be sent/uploaded every time.

1.1.6 Authorization

Info

- **Authorization** answers question: "What are you allowed to do?" => By using Authorities & Roles
(Defines access to Controller Endpoints to which Authorities & Roles are assigned)
- **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.

1.1.7 Authorization - Authorities vs Roles

Info

- Spring Security is based on **Authorities**.

User is first Authenticated through combination of its Username and Password.

Then Spring creates User Object that contains list of Authorities that are assigned to that user (from DB for instance).

These Authorities can then be used inside the Controller to control access to the endpoints.

- 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')")`.

- Roles can be used on their own to control access to endpoints.

Alternatively **Roles** can be used as **containers for Authorities**.

Inside DB you assign Roles to Users and then you assign Authorities to these Roles (through @ManyToMany Relationship)

Then when creating User Object you load it with Authorities that are related to Roles of Authenticated User.

- You can also **add Roles to list of User Authorities**.

This might be useful to detect from which Roles those Authorities came from.

For instance both ADMIN and USER Roles might have `book.read` Authority

- But if `book.read` Authority came from ADMIN Role then Admin can read any book (so no other checks are necessary).
- But if `book.read` Authority came from USER Role then User can only read his own books.

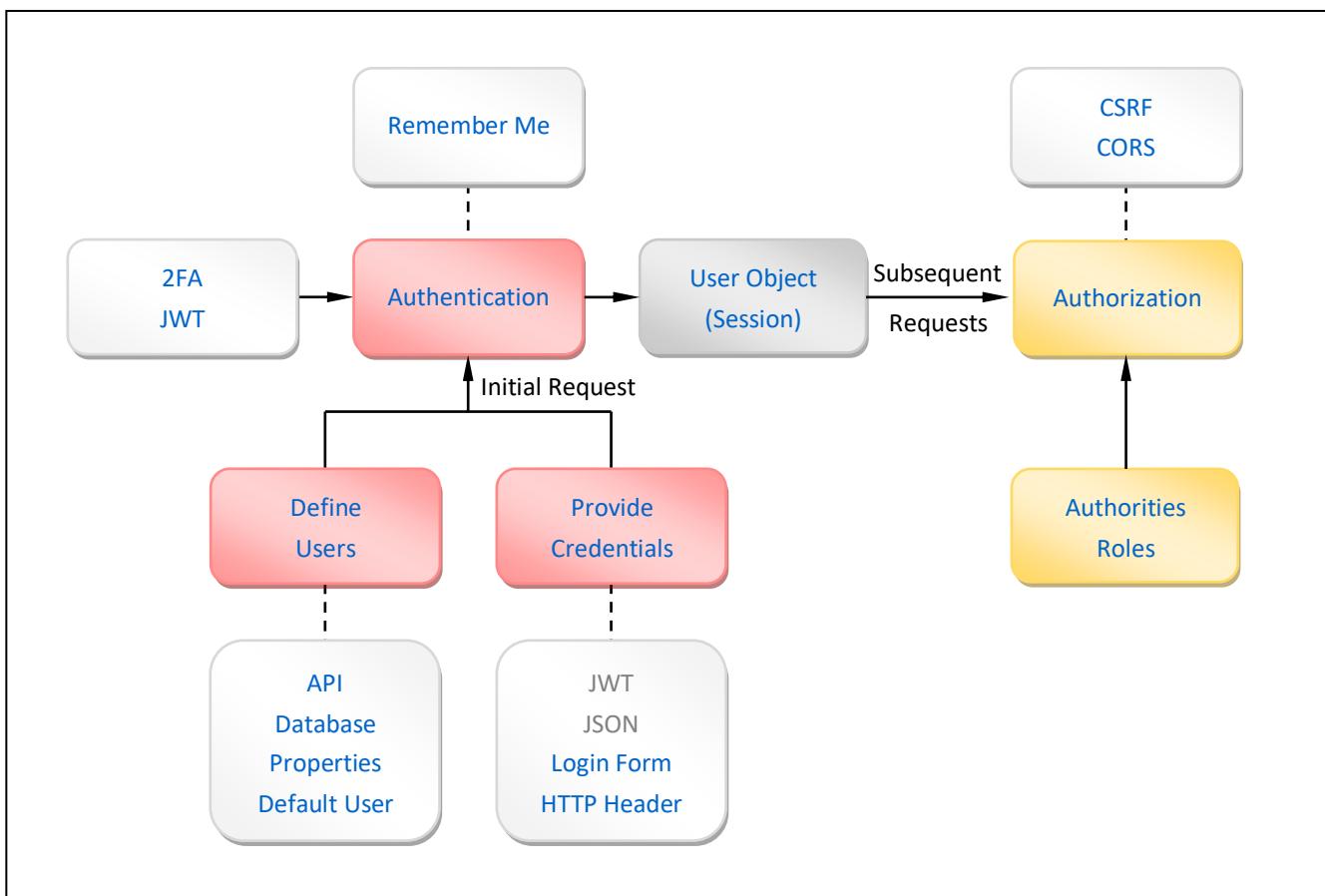
In that case an additional check is needed through **Custom Authorization** where we compare ID of current User with User ID assigned to the Book, and allow access to endpoint only if these two IDs match (if User owns the Book).

1.1.8 Security - Different Implementations

Info

- Below Schema show different Security Implementations covered in this book and how they relate to each other.

Schema



1.1.9 Security - WebSecurityConfig.java

Info

- Class `WebSecurityConfig.java` is going to be a central point in specifying different settings related to Security.\
- So here is a short overview of different things that can be configured.

WebSecurityConfig.java

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

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;

@Configuration
@EnableWebSecurity
@RequiredArgsConstructor
@EnableGlobalMethodSecurity(securedEnabled = true, prePostEnabled = true) //Enables @Secured & @PreAuthorize
public class WebSecurityConfig extends WebSecurityConfigurerAdapter {

    private final UserDetailsService userDetailsService;

    @Bean
    PasswordEncoder passwordEncoder() {
        return NoOpPasswordEncoder.getInstance();
    }

    //=====
    // CONFIGURE
    //=====

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

        //SPECIFY ACCESS TO ENDPOINTS
        httpSecurity.authorizeRequests().antMatchers("/Authenticate").permitAll(); //Anonymous Access (no Login)
        httpSecurity.authorizeRequests().antMatchers("/Hello").hasRole("USER"); //Authenticated Access
        httpSecurity.authorizeRequests().anyRequest().authenticated(); //Authenticated Access

        //DEFAULT LOGIN FORM
        httpSecurity.formLogin();

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

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

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

    }
}
```

MyController.java

```
@Secured({ "ROLE_ADMIN", "ROLE_USER" })
@PreAuthorize("hasAnyRole('ADMIN', 'USER')")
```

1.1.10 Security - MyUserDetailsService.java

Info

- MyUserDetailsService.java returns **User Object** with Username, **Password** and **Authorities** (from entered Username).
For instance you can use entered Username to get Account from DB and use it to create User Object.
Spring will compare entered Password with Password that you place inside returned User Object.
- Returned User Object must Implement **UserDetails Interface** so you can use either
 - built-in User Object or
 - your Custom Object that Implements UserDetails Interface
- Following tutorials show this
 - [Manually Create User Object](#)
 - [Define Users in DB](#)

MyUserDetailsService.java

(from Manually Create User Object)

```
@Service
public class MyUserDetailsService implements UserDetailsService {

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

        //HARD CODED USER
        String username = "myuser";
        String password = "mypassword";
        String role      = "ROLE_USER";

        //CHECK USERNAME
        if (!username.equals(enteredUsername)) {
            throw new UsernameNotFoundException(enteredUsername);
        }

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

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

        //RETURN USER
        return user;
    }
}
```

```
@Service
public class MyUserDetailsService implements UserDetailsService {

    @Autowired
    AccountRepository accountRepository;

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

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

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

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

        //RETURN USER
        return user;
    }

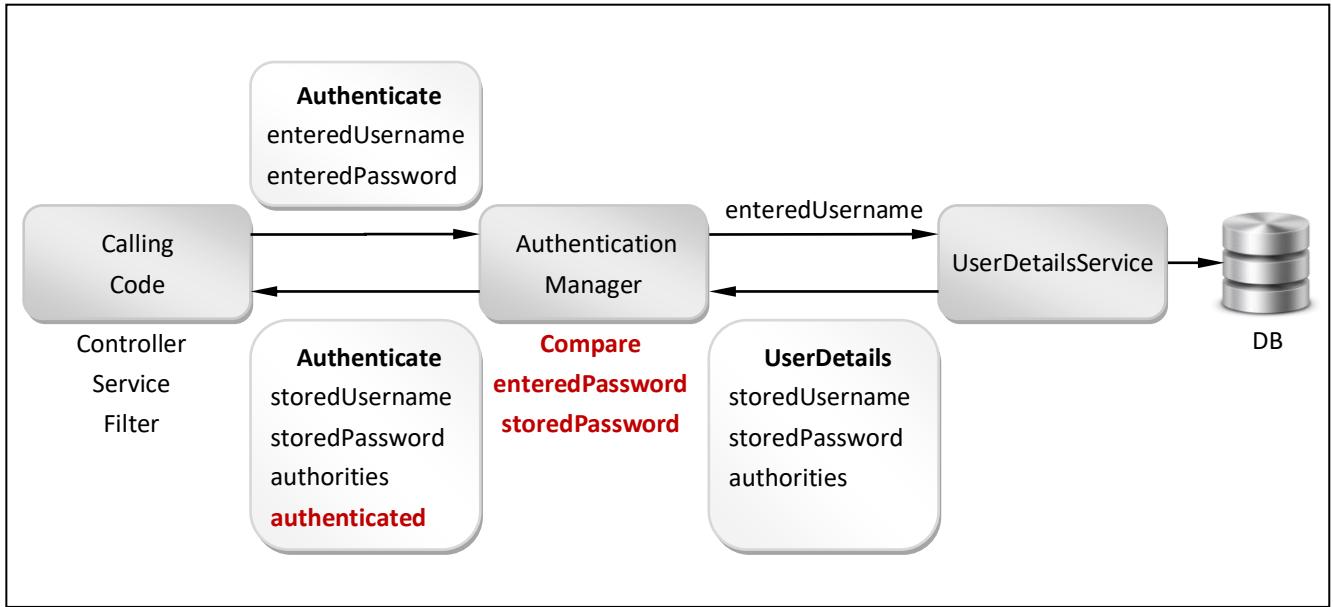
}
```

1.1.11 Security - UserDetailsService vs AuthenticationManager

Info

- **UserDetailsService** uses **enteredUsername** to return **UserDetails** Object with **storedPassword** & **authorities**
- **AuthenticationManager** uses **enteredUsername** & **enteredPassword** to return **Authenticate** Object if passwords match
- **UserDetails** Object is DTO used to transfer Username, Password & Authorities from DB into **Authenticate** Object
- **Authenticate** Object is used by Spring Security to control access to Endpoints using **authorities** and **authenticated**.

Authentication Process



UserDetailsService

- **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**
 - 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.

AuthenticationManager

- **AuthenticationManager** is used to compare
 - **enteredPassword** with the
 - **storedPassword** in **UserDetails** Object (that was returned by **UserDetailsService**)
- **AuthenticationManager**
 - is called with **Authentication** Object as Input Parameter that contains **enteredUsername** and **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** (also taken from **UserDetails** Object)
 - **authenticated = true**
- Returned **Authentication** Object is accepted by the code that called **AuthenticationManager**. That calling code can now
 - store **Authentication** Object into Context/Session
 - return JWT token

UserDetails vs Authenticate

- At the end of the day **UserDetails** and **Authenticate** Objects might contain same data: Username, Password, Authorities. But it is the **Authenticate** Object from which Spring Security will use **Authorities** to control access to Restricted Resources. Since **Authenticate** Object also has **Boolean authenticated** which must be true for Spring to even look at Authorities.
- **UserDetails** Object as DTO was just used as a temporary storage for Username, Password and Authorities as they make their way from Database into **Authenticate** Object.
- But **UserDetails** Object can also contain some additional User data that are not used for Authorization and will not be transferred to **Authenticate** Object. In that case it makes sense to have them both inside Context.

1.2 Define Users

Info

- Following tutorials show different ways to [Define Users](#).

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, LOADER
```

Users defined in API or DB

- With Users defined in API or DB you can have **many Users** with different Username, Password and Role.
- Using this setup you are good to go with **Role Based Security**.
- Additionally for each Role you could specify related Authorities.
This way you can implement **Authority Based Security**.

1.2.1 Default User

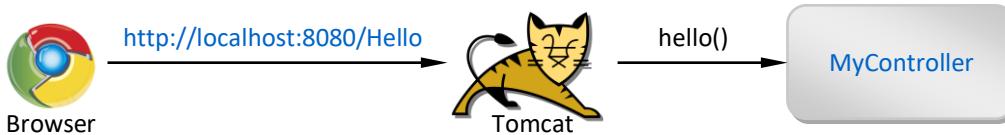
Info

[G]

- This tutorial shows how to use **Default User** that has
 - Username: **user**
 - Password: (**randomly generated** when you start Application, it is displayed in the Console)

Application Schema

[Results]



Spring Boot Starters

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

Procedure

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

MyController.java

```
package com.ivoronline.springboot_security_default.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 hello() {
        return "Hello from Controller";
    }
}
```

Results

- <http://localhost:8080>Hello>
- You get redirected to <http://localhost:8080/login>
 - Username: user
 - Password: **506e6f00-2b11-4036-96d6-74633e94da2d** (copy from Console output)
 - Sign in
- JSESSIONID Cookie is returned and stored in your Browser
- You get redirected back to <http://localhost:8080>Hello>
- <http://localhost:8080/logout>
 - Log Out

Console

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

<http://localhost:8080/login>

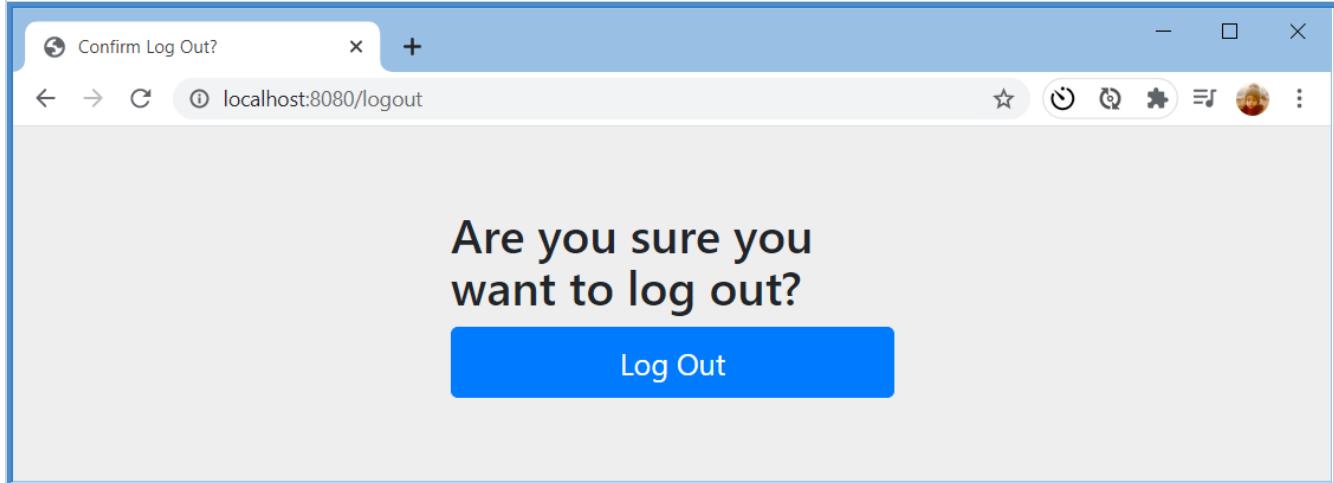
(JSESSIONID Cookie is stored in the Browser)

The screenshot shows a browser window with a "Please sign in" form. The form has two input fields: one for "user" containing "user" and another for a password containing several dots. Below the fields is a blue "Sign in" button. At the bottom of the browser window, the developer tools Network tab is open, showing a table of cookies. One cookie is selected: "JSESSIONID" with a value of "71FFF6A3E48188B4C4AFB3F55CB2662F". The table includes columns for Name, Value, Domain, Path, Expires / Max-Age, Size, Http..., Secure, Sam..., and Priority.

Name	Value	Domain	Path	Expires / Max-Age	Size	Http...	Secure	Sam...	Priority
JSESSIONID	71FFF6A3E48188B4C4AFB3F55CB2662F	localhost	/	Session	42	✓			Medi...

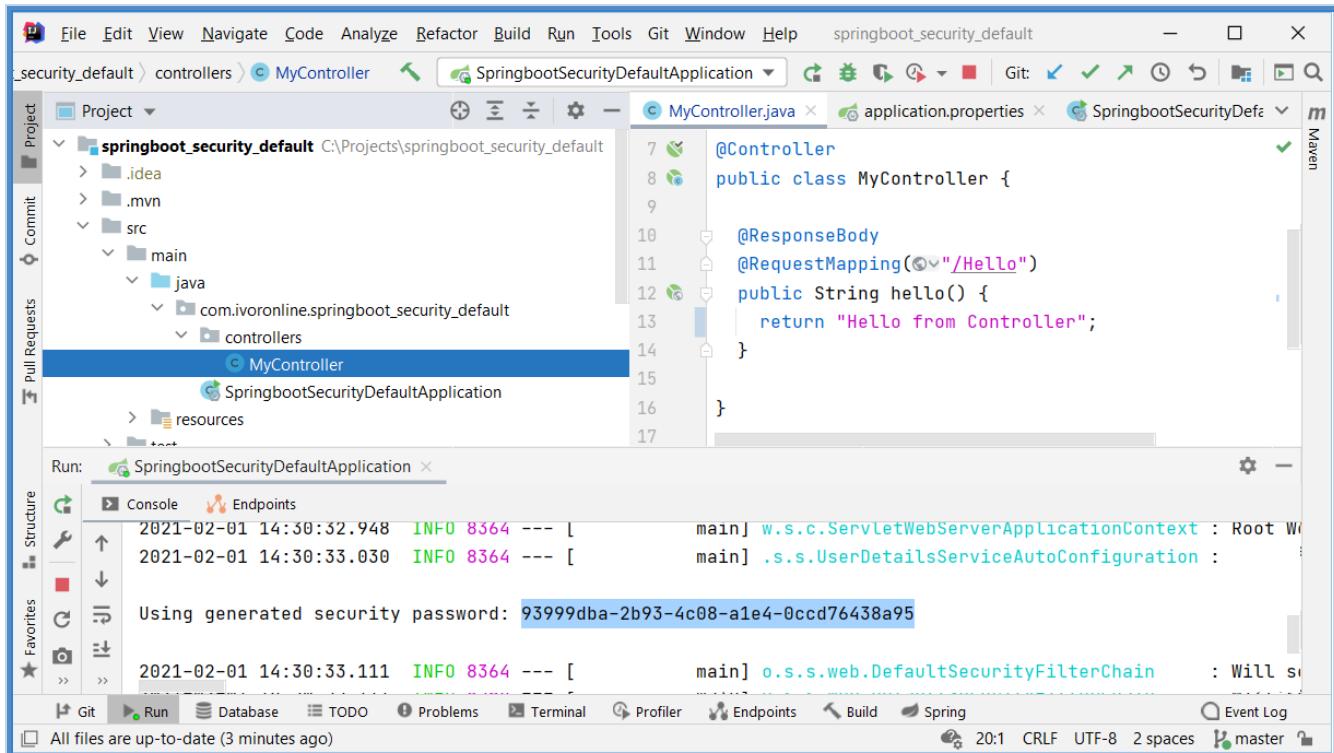
Redirects to <http://localhost:8080>Hello>

The screenshot shows a browser window with the URL "localhost:8080>Hello" in the address bar. The main content area displays the text "Hello from Controller".



Application Structure

(Temporary Code is displayed in the Console)



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>
```

1.2.2 application.properties

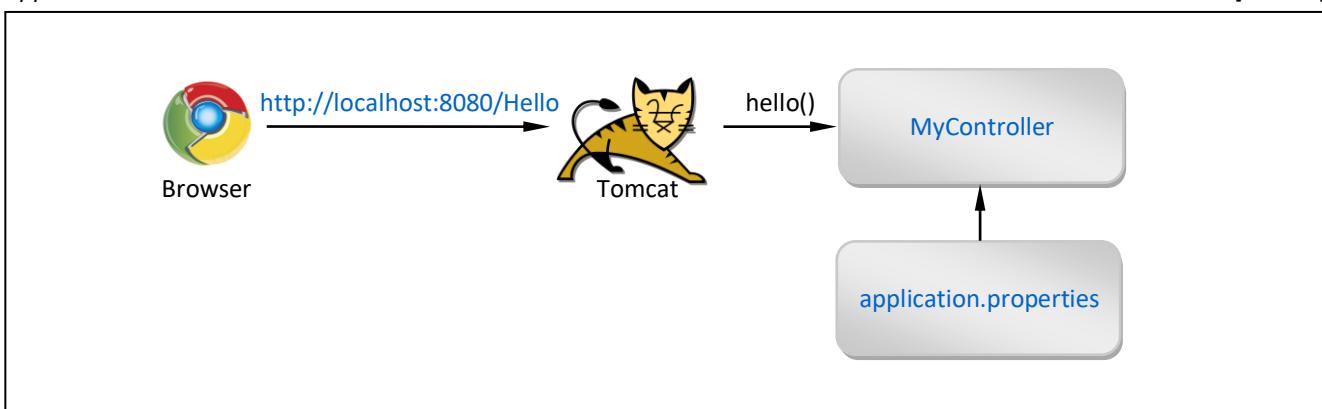
Info

[G]

- This tutorial shows how to use [application.properties](#) to specify single custom User (you can't specify multiple Users).

Application Schema

[Results]



Spring Boot Starters

GROUP	DEPENDENCY	DESCRIPTION
Web	Spring Web	Enables @RequestMapping. Includes Tomcat HTTP Server.
Security	Spring Security	Enables Spring Security.

Procedure

- Create Project:** `springboot_security_default` (add Spring Boot Starters from the table)
- Edit File:** `application.properties` (contains Role, User, Password)
- 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 = mypassword
spring.security.user.roles     = USER
```

MyController.java

```
package com.ivoronline.springboot_security_user_applicationproperties.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 hello() {
        return "Hello from Controller";
    }
}
```

Results

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

<http://localhost:8080/login>

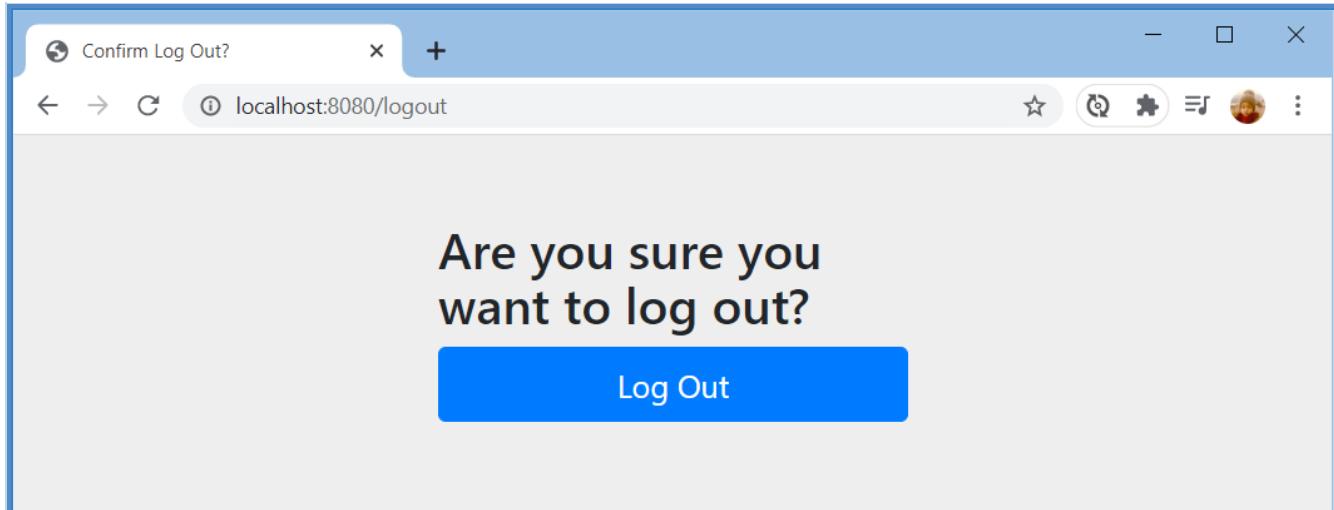
(JSESSIONID Cookie is stored in the Browser)

The screenshot shows a browser window with a "Please sign in" form. The "Username" field contains "myuser" and the "Password" field contains ".....". A blue "Sign in" button is at the bottom. Above the browser is a note: "(JSESSIONID Cookie is stored in the Browser)". Below the browser is a screenshot of the developer tools Application tab. The Cookies section shows a single entry for the domain "http://localhost:8080":

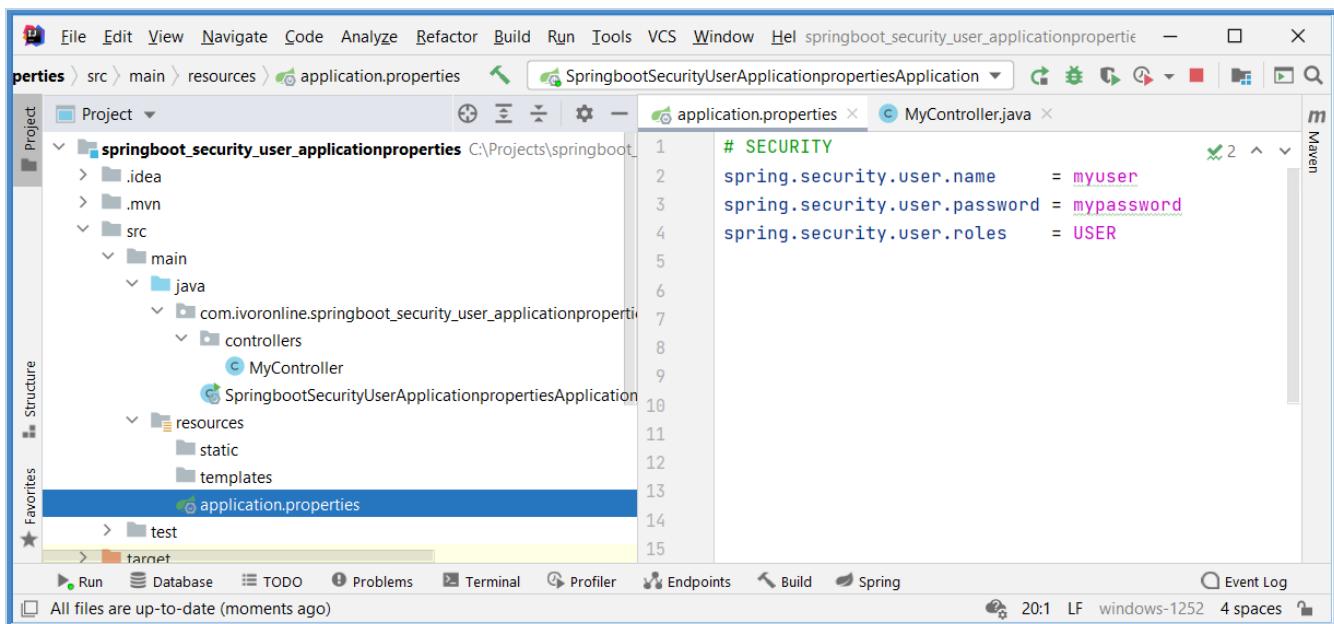
Name	Value	Domain	Path	Expires / Ma...	S...	Htt...	Sec...	Sa...	Pri...
JSESSI...	89CA1E063E2E2525697EC75...	localhost	/	Session	42	✓			Me...

Redirects to <http://localhost:8080>Hello>

The screenshot shows a browser window with the URL "localhost:8080/Hello". The page content is "Hello from Controller".



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>
```

1.2.3 API - userDetailsService()

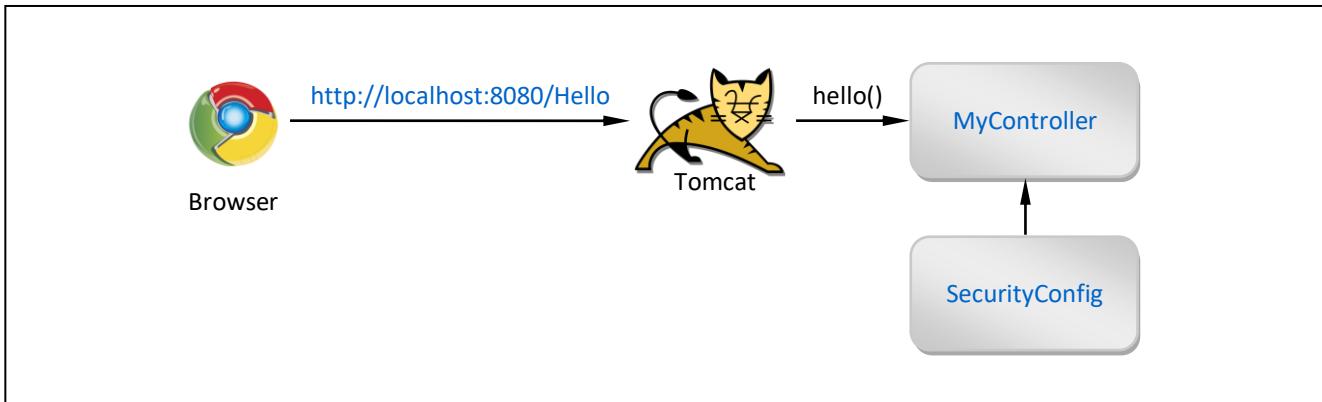
Info

[G]

- This tutorial shows how to use API to specify multiple Users by (application.properties can only specify single User)
 - extending [WebSecurityConfigurerAdapter](#)
 - overriding Method [userDetailsService\(\)](#)

Application Schema

[Results]



Users

USERNAME	PASSWORD	ROLES
myadmin	myadminpassword	ADMIN
myuser	myuserpassword	USER

Spring Boot Starters

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

Procedure

- Create Project: `springboot_security_class` (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: `SecurityConfig.java` (inside config package)

MyController.java

```
package com.ivoronline.springboot_security_class.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 hello() {
        return "Hello from Controller";
    }

}
```

SecurityConfig.java

```
package com.ivoronline.springboot_security_class.config;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
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.provisioning.InMemoryUserDetailsManager;

@Configuration
@EnableWebSecurity
public class SecurityConfig extends WebSecurityConfigurerAdapter {

    @Bean
    @Override
    protected UserDetailsService userDetailsService() {

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

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

        return new InMemoryUserDetailsManager(admin, user);
    }
}
```

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)

The screenshot shows a browser window with a 'Please sign in' form. The form has two input fields: one containing 'myuser' and another containing '.....'. Below the fields is a blue 'Sign in' button. Above the form, the browser title bar says 'Please sign in' and the address bar shows 'localhost:8080/login'. The browser interface includes standard navigation buttons and a toolbar.

Below the browser window is the developer tools interface, specifically the Application tab. It shows a table of cookies. One cookie is highlighted:

Name	Value	Domain	Path	Expires / Max-Age	Size	Http...	Secure	Same...	Priority
JSESSIONID	51D4840DF1A5B4B9148AE6D8D798F2E3	localhost	/	Session	42	✓			Medi...

The Cache section of the developer tools shows a single entry: '51D4840DF1A5B4B9148AE6D8D798F2E3'.

Redirects to <http://localhost:8080>Hello>

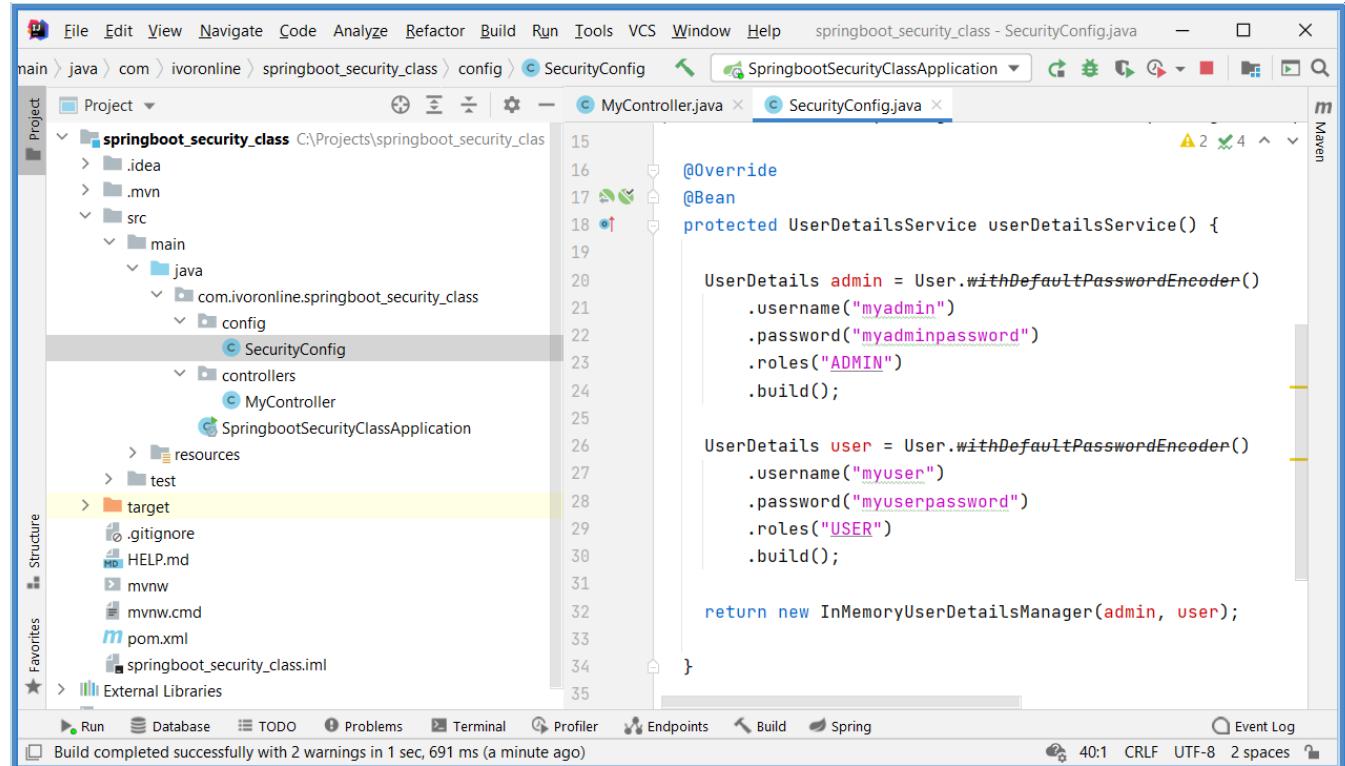
The screenshot shows a browser window with the address bar showing 'localhost:8080/hello'. The main content area displays the message 'Message from Controller'.

<http://localhost:8080/logout>

(Redirects to <http://localhost:8080/login>)

The screenshot shows a browser window with a confirmation dialog titled 'Confirm Log Out?'. The dialog contains the text 'Are you sure you want to log out?' and a large blue 'Log Out' button.

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>
```

1.2.4 API - configure()

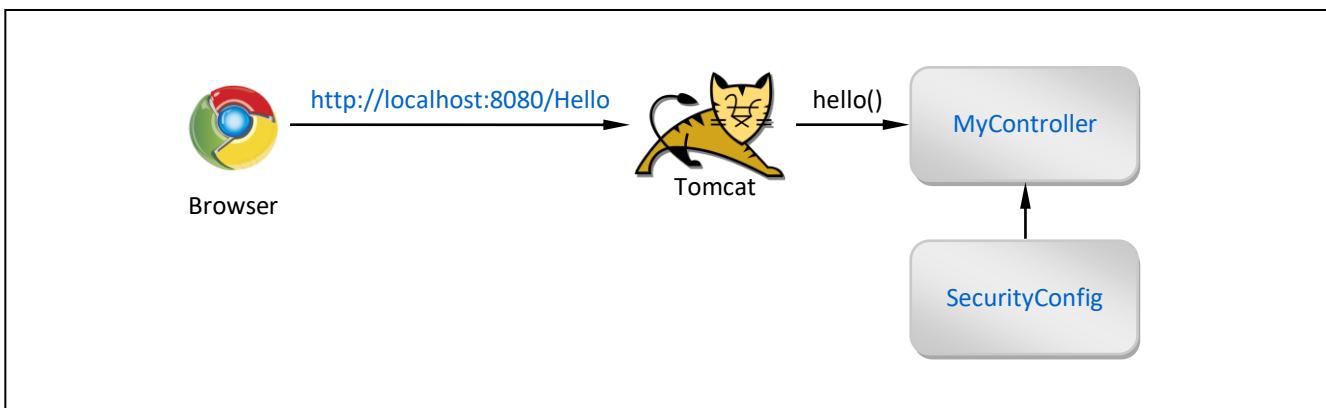
Info

[G]

- This tutorial shows how to use API to specify multiple Users by (application.properties can only specify single User)
 - extending [WebSecurityConfigurerAdapter](#)
 - overriding Method `configure()` (AuthenticationManagerBuilder auth)
- {noop}** stands for No-Operation and it instructs Spring to store password as is (without encryption).

Application Schema

[Results]



Users

USERNAME	PASSWORD	ROLES
myadmin	myadminpassword	ADMIN
myuser	myuserpassword	USER

Spring Boot Starters

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

Procedure

- **Create Project:** `springboot_security_class2` (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:** `SecurityConfig.java` (inside config package)

MyController.java

```
package com.ivoronline.springboot_security_class2.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 hello() {
        return "Hello from Controller";
    }

}
```

SecurityConfig.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.web.configuration.EnableWebSecurity;
import org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;

@Configuration
@EnableWebSecurity
public class SecurityConfig extends WebSecurityConfigurerAdapter {

    @Override
    protected void configure(AuthenticationManagerBuilder auth) throws Exception {

        //ADD ADMIN
        auth.inMemoryAuthentication()
            .withUser("myadmin")
            .password("{noop}myadminpassword")
            .roles("ADMIN");

        //ADD USER
        auth.inMemoryAuthentication()
            .withUser("myuser")
            .password("{noop}myuserpassword")
            .roles("USER");
    }

}
```

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>

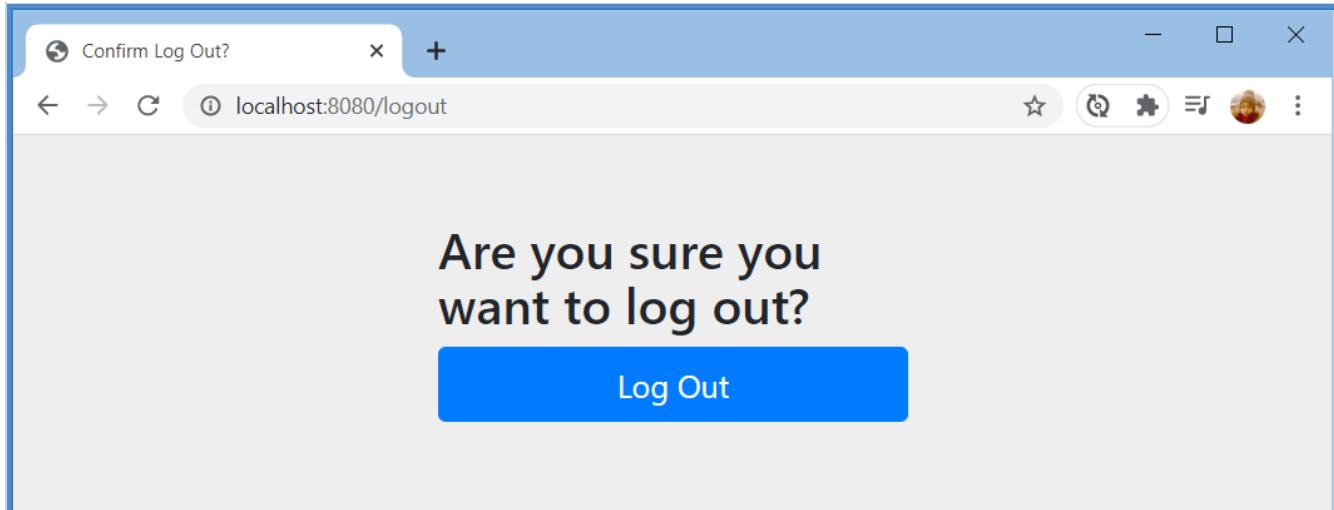
(JSESSIONID Cookie is stored in the Browser)

The screenshot shows a browser window with a "Please sign in" form. The "Username" field contains "myuser" and the "Password" field contains "myuserpassword". A large blue "Sign in" button is at the bottom. Above the browser is a note: "(JSESSIONID Cookie is stored in the Browser)". Below the browser is a screenshot of the developer tools Application tab. The Cookies section shows a table with one row:

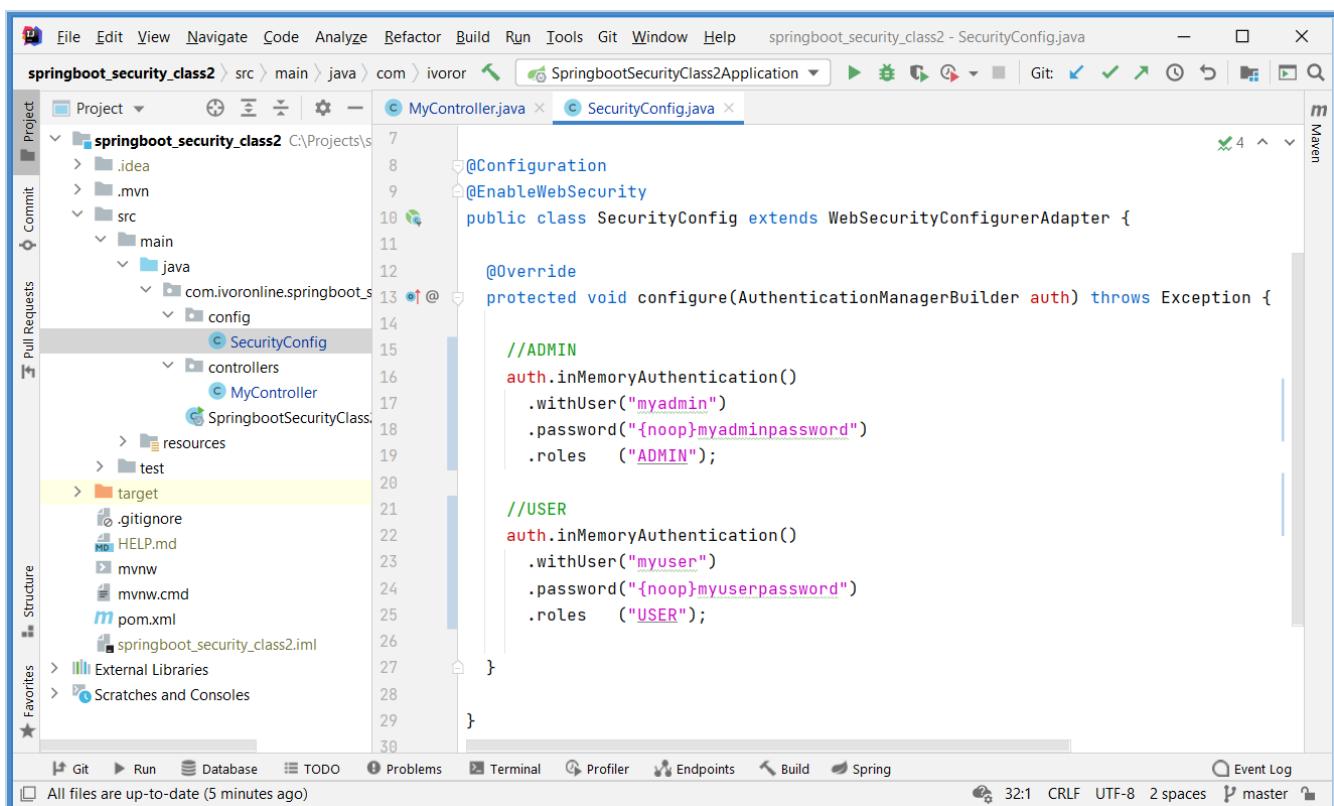
Name	Value	Domain	Path	Expires / Ma...	S...	Htt...	Sec...	Sa...	Pri...
JSESSI...	89CA1E063E2E2525697EC75...	localhost	/	Session	42	✓			Me...

Redirects to <http://localhost:8080>Hello>

The screenshot shows a browser window with the URL "localhost:8080>Hello". The page content is "Hello from Controller".



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>
```

1.2.5 DB

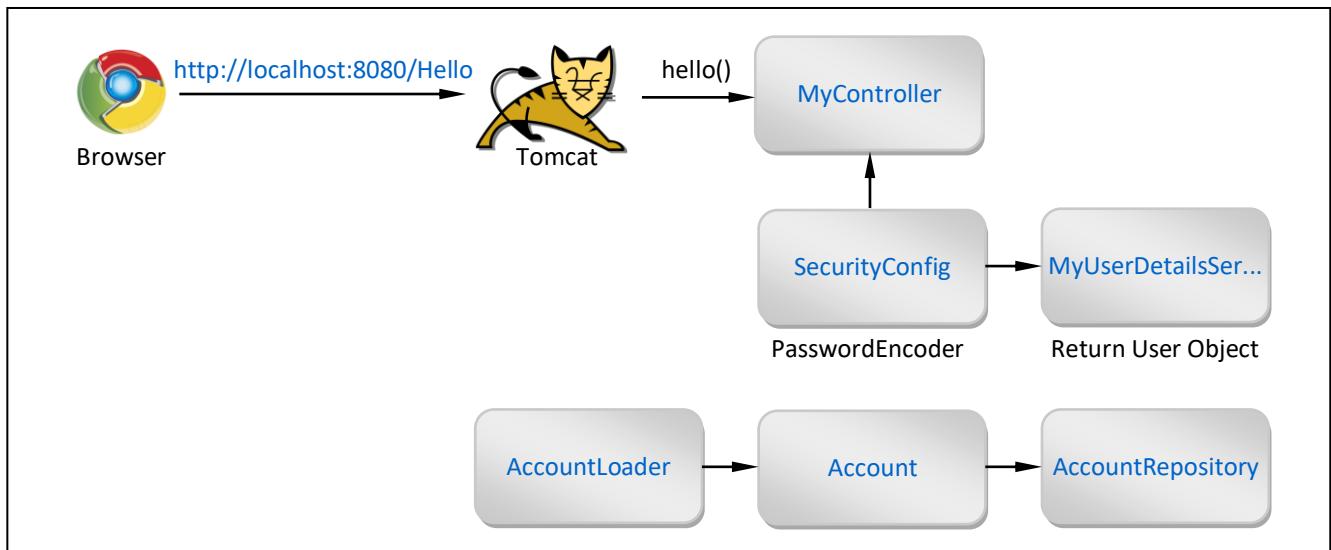
Info

[G]

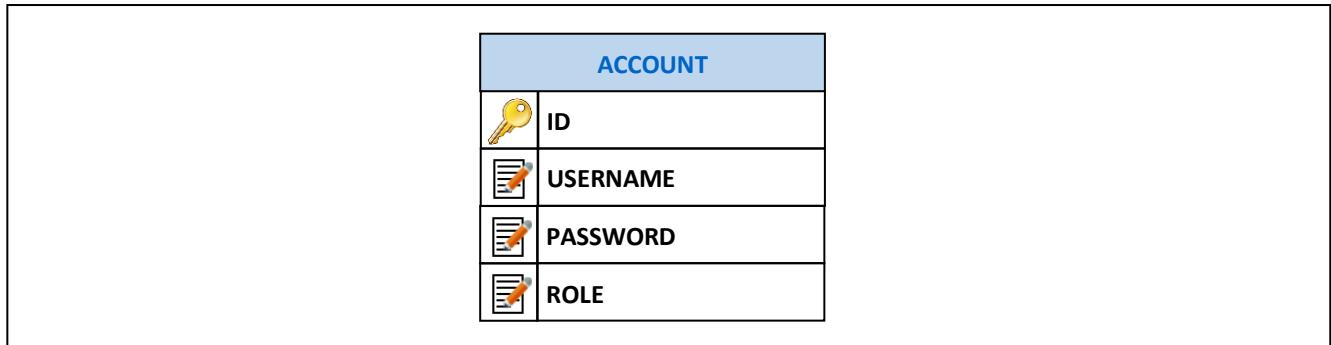
- This tutorial shows how to define/store Users in DB.
- Users will be stored in the **ACCOUNT** Table for which we will create **Account** Entity.
- Account Entity will have: Username, Password and Role.
- We are using **Account** Entity to avoid confusion with built-in **User** Class .

Application Schema

[Results]



DB Schema



Account

(Loaded Data)

ID	USERNAME	PASSWORD	ROLE
1	john	johnpassword	ROLE_ADMIN

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	H2 Database	Enables in-memory H2 Database

MyUserDetailsService.java

- If you
 - declare Class that `implements UserDetailsService` (it is called Service because it SERVES UserDetails Object)
 - and `@Override` its Method `loadUserByUsername(String username)`
 - then this Method will be called when User enters Username and Password through Login Form
- Method
 - should return Object of Class that Implements `UserDetails` Interface
 - which should contain: Username, **Password**, Authorities
 - which in our case will be built-in `User Class`
- Method Body is completely customizable
 - so we take Account from DB based on `username` Input Parameter
 - and from that Account we create and return `User Object`
- This User Object
 - will contain **Password from DB**
 - which Spring will compare with the **Password from Login Form**
 - and if they match Spring will **Authenticate User** by setting User Object Property `enabled = true`

MyUserDetailsService.java

(Returns User Object with: Username, **Password**, Authorities)

```
@Service
public class MyUserDetailsService implements UserDetailsService {

    @Override
    public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {
        // Use DB Account to return User Object that contains: Username, Password, Authorities
    }

}
```

SecurityConfig.java

- SecurityConfig.java is needed only to specify `PasswordEncoder`.
- If you don't need access to H@ Console then `configure()` is not necessary.

SecurityConfig.java

```
public class SecurityConfig extends WebSecurityConfigurerAdapter {

    @Bean
    PasswordEncoder passwordEncoder() {
        return NoOpPasswordEncoder.getInstance();
    }
}
```

Procedure

- Create Project: `springboot_dbauthentication` (add Spring Boot Starters from the table)
- Edit File: `application.properties` (specify H2 DB name, enable H2 Web Console)
- Create Package: `config` (inside main package)
 - Create Class: `AccountLoader.java` (inside package config)
 - Create Class: `SecurityConfig.java` (inside package config)
- Create Package: `entities` (inside main package)
 - Create Class: `Account.java` (inside package entities)
- Create Package: `repositories` (inside main package)
 - Create Interface: `AccountRepository.java` (inside package repositories)
- Create Package: `services` (inside main package)
 - Create Class: `MyUserDetailsService.java` (inside package services)
- Create Package: `controllers` (inside main package)
 - Create Class: `MyController.java` (inside package controllers)

application.properties

```
# H2 CONSOLE
spring.h2.console.enabled = true
spring.datasource.url      = jdbc:h2:mem:testdb
```

Account.java

```
package com.ivoronline.springboot_dbauthentication.entities;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;

@Entity
public class Account {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    public Integer id;
    public String username;
    public String password;
    public String role;

}
```

AccountRepository.java

```
package com.ivoronline.springboot_dbauthentication.repositories;

import com.ivoronline.springboot_dbauthentication.entities.Account;
import org.springframework.data.jpa.repository.JpaRepository;

public interface AccountRepository extends JpaRepository<Account, Integer> {
    Account findByUsername(String Username);
}
```

AccountLoader.java

```
package com.ivorononline.springboot_dbauthentication.config;

import com.ivorononline.springboot_dbauthentication.entities.Account;
import com.ivorononline.springboot_dbauthentication.repositories.AccountRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.CommandLineRunner;
import org.springframework.stereotype.Component;
import org.springframework.transaction.annotation.Transactional;

@Component
public class AccountLoader implements CommandLineRunner {

    @Autowired private AccountRepository accountRepository;

    @Override
    @Transactional
    public void run(String... args) throws Exception {

        //CREATE ACCOUNT
        Account account = new Account();
        account.username = "john";
        account.password = "johnpassword";
        account.role = "ROLE_ADMIN";

        //STORE ACCOUNT INTO DB
        accountRepository.save(account);
    }
}
```

MyUserDetailsService.java

```
package com.ivorononline.springboot_dbauthentication.services;

import com.ivorononline.springboot_dbauthentication.entities.Account;
import com.ivorononline.springboot_dbauthentication.repositories.AccountRepository;
import org.springframework.beans.factory.annotation.Autowired;
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.List;

@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.getPassword();

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

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

        //RETURN USER
        return user;
    }
}
```

SecurityConfig.java

```
package com.ivoronline.springboot_dbauthentication.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 SecurityConfig extends WebSecurityConfigurerAdapter {

    @Bean
    PasswordEncoder passwordEncoder() {
        return NoOpPasswordEncoder.getInstance();
    }

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

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

        //RESTRICT ACCESS TO EVERYTHING ELSE (BEHIND LOGIN FORM)
        httpSecurity
            .authorizeRequests().anyRequest().authenticated()
            .and().formLogin()
            .and().httpBasic();

    }

}
```

MyController.java

```
package com.ivoronline.springboot_dbauthentication.controllers;

import com.ivoronline.springboot_dbauthentication.repositories.AccountRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.ResponseBody;

@Controller
public class MyController {

    @Autowired
    AccountRepository accountRepository;

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

}
```

Results

- <http://localhost:8080>Hello>
- You get redirected to <http://localhost:8080/login>
 - Username: **john**
 - Password: **johnpassword**
 - Sign in
- You get redirected back to <http://localhost:8080>Hello>

<http://localhost:8080/login> - john - johnpassword

(JSESSIONID Cookie is stored in the Browser)

The screenshot shows a browser window with a login form titled "Please sign in". The username field contains "john" and the password field contains ".....". A blue "Sign in" button is at the bottom. Above the browser is a status bar with the URL "localhost:8080/login". Below the browser is the Chrome DevTools Application tab, which displays a table of cookies. One cookie is selected: "Name": "JSESSIONID", "Value": "4CA1E133CE2694C20716B0A3C750053E", "Domain": "localhost", "Path": "/", "Expires / ...": "Session", "Secure": true, "HttpOnly": true, "SameSite": null, "Priority": "Me...".

Name	Value	Domain	P...	Expires / ...	Si...	Htt...	Sec...	Sam...	Prio...
JSESSIONID	4CA1E133CE2694C20716B0A3C750053E	localhost	/	Session	42	✓			Me...

Redirects to <http://localhost:8080>Hello>

The screenshot shows a browser window with the URL "localhost:8080>Hello". The page content is "Hello from Controller". Above the browser is a status bar with the URL "localhost:8080>Hello".

The screenshot shows the H2 Console interface. On the left, there's a sidebar with a tree view of databases and schemas. The current connection is 'jdbc:h2:mem:testdb'. The main area contains a SQL editor with the query 'SELECT * FROM ACCOUNT' and its results:

```
SELECT * FROM ACCOUNT;
+----+-----+-----+-----+
| ID | PASSWORD | ROLE    | USERNAME |
+----+-----+-----+-----+
| 1  | johnpassword | ROLE_ADMIN | john      |
+----+-----+-----+-----+
(1 row, 6 ms)
```

Below the results is a 'Edit' button.

Application Structure

The screenshot shows the IntelliJ IDEA IDE. The left side displays the project structure for 'springboot_dbauthentication'. The 'src/main/java/com.ivoronline.springboot_dbauthentication/services' package is selected, showing the 'AccountService.java' file. The code editor shows the following Java code:

```
@Service
public class AccountService implements UserDetailsService {

    @Autowired
    AccountRepository accountRepository;

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

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

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

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

        //RETURN USER
        return user;
    }
}
```

The code editor has syntax highlighting and line numbers. The bottom status bar indicates the code is up-to-date.

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-data-jpa</artifactId>
    </dependency>

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

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

</dependencies>
```

1.3 Authentication

Info

- Following tutorials show different ways in which User can provide his **Credentials (Username & Password)** either through
 - [Login Form](#) ([Default](#), [Custom](#))
 - [Authentication Header](#) ([Postman - Add Header](#), [Postman - Generate Header](#))
- When you access any endpoint for the first time (after Application start) you will be redirected to default [Login Form](#).
Upon successful Login (they allow subsequent HTTP Requests without logging in again)
 - [Session Object](#) is created on the Server (stores User data)
 - [JSESSIONID Cookie](#) is stored in your Browser (points to that Session, it is sent with every subsequent HTTP Request)

Login Form

- When User tries to access restricted endpoint it is redirected to Login Form where it can enter Username and Password.
- Spring then tries to match entered combination of Username and Password against stored Users.
- If it finds User with entered Username and Password it creates User Object that contains that Username and Password.

Authentication Header

- HTTP Request for Endpoint can have Authentication Header containing Username and Password.
- In that case Login Form is not displayed since Username & Password are already provided through Authentication Header.
- After extracting Username & Password from the Authentication Header Spring proceeds as described in [Login Form](#).

1.3.1 Automatic - Login Form - Default

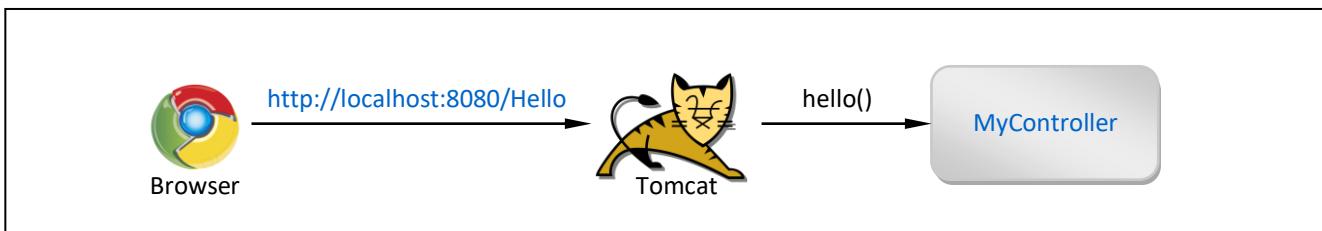
Info

[G]

- This tutorial shows how to allow User to provide Username and Password through default Login Form.
User is stored in [application.properties](#) (these two tutorials are exactly the same since they both use default Login Form).

Application Schema

[Results]



Spring Boot Starters

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

Procedure

- Create Project:** `springboot_security_default` (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)

application.properties

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

MyController.java

```
package com.ivoronline.springboot_security_default.controllers;

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

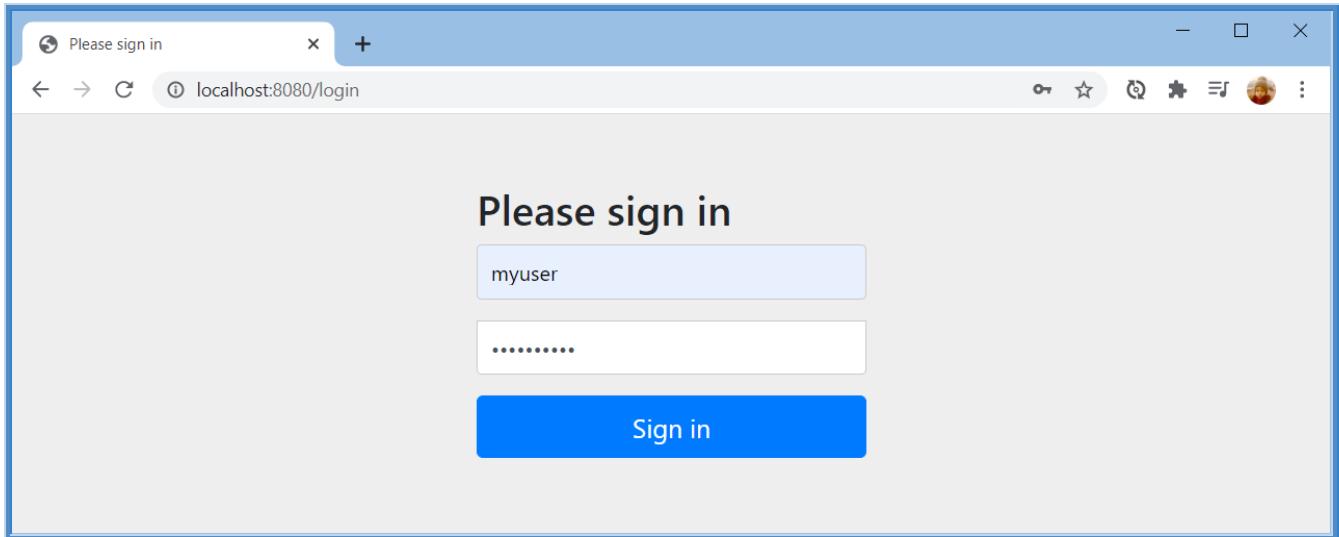
@Controller
public class MyController {

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

Results

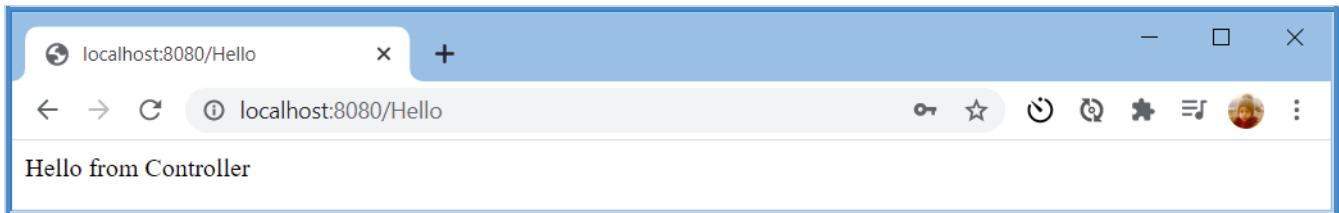
- <http://localhost:8080>Hello>
- You get redirected to <http://localhost:8080/login>
 - Username: **myuser**
 - Password: **mypassword**
 - Sign in
- You get redirected back to <http://localhost:8080>Hello>
- Logout Form <http://localhost:8080/logout>
 - Log Out

<http://localhost:8080/login>

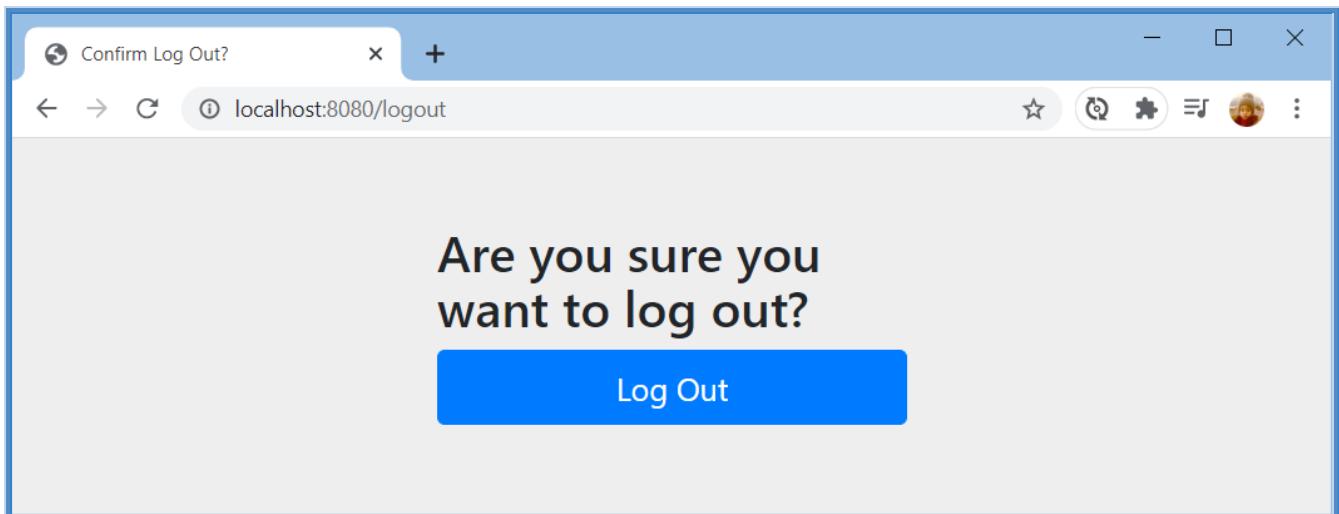


Please sign in

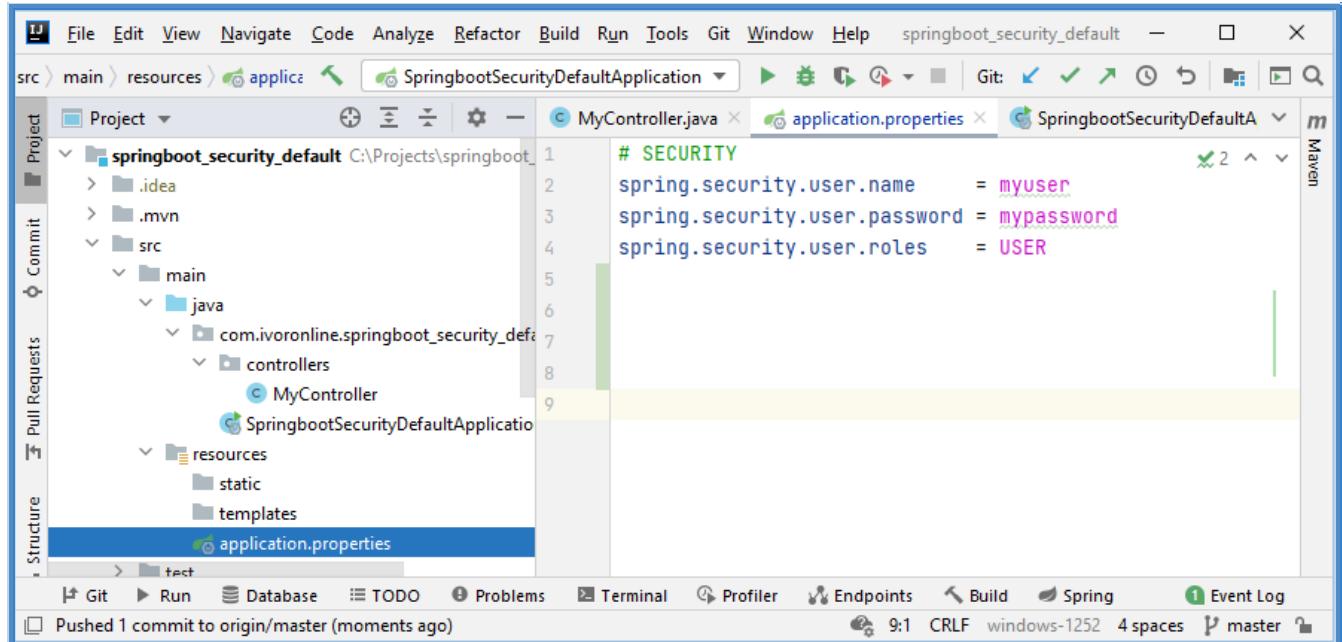
Redirects to <http://localhost:8080>Hello>



<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>
```

1.3.2 Automatic - Login Form - Custom

Info

[G]

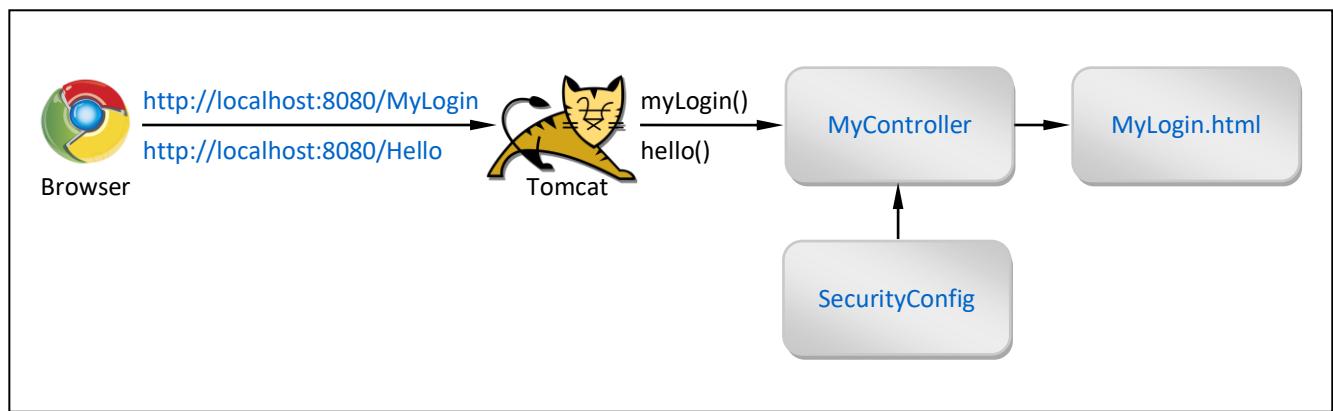
- This tutorial shows how to allow User to provide Username and Password through Custom Login Form.
- Which **HTML Page** to use as Custom Login Form is defined in `SecurityConfig.java` which must extend `WebSecurityConfigurerAdapter` and `@Override configure()` Method.
- And since we are now providing our own Implementation we need to add additional configuration to allow access to Endpoints `httpSecurity.authorizeRequests().antMatchers("/Hello").hasRole("USER")`.

SecurityConfig.java

```
public class SecurityConfig extends WebSecurityConfigurerAdapter {  
    @Override protected void configure(HttpSecurity httpSecurity) throws Exception {  
        httpSecurity.formLogin().loginPage("/MyLogin").loginProcessingUrl("/login");  
        ...  
    }  
}
```

Application Schema

[Results]



Spring Boot Starters

GROUP	DEPENDENCY	DESCRIPTION
Web	Spring Web	Enables <code>@RequestMapping</code> and Tomcat
Security	Spring Security	Enables Spring Security
Template Engines	Thymeleaf	Enables Controller to return reference to HTML Page <code>MyLogin.html</code>

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: `SecurityConfig.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 = mypassword
spring.security.user.roles     = USER
```

MyController.java

```
package com.ivoronline.springboot_loginform_custom.controllers;

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

@Controller
public class MyController {

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

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

SecurityConfig.java

```
package com.ivorononline.springboot_security_loginform_custom.config;

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;

@Configuration
@EnableWebSecurity
public class SecurityConfig extends WebSecurityConfigurerAdapter {

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

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

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

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

    }

}
```

MyLogin.html

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

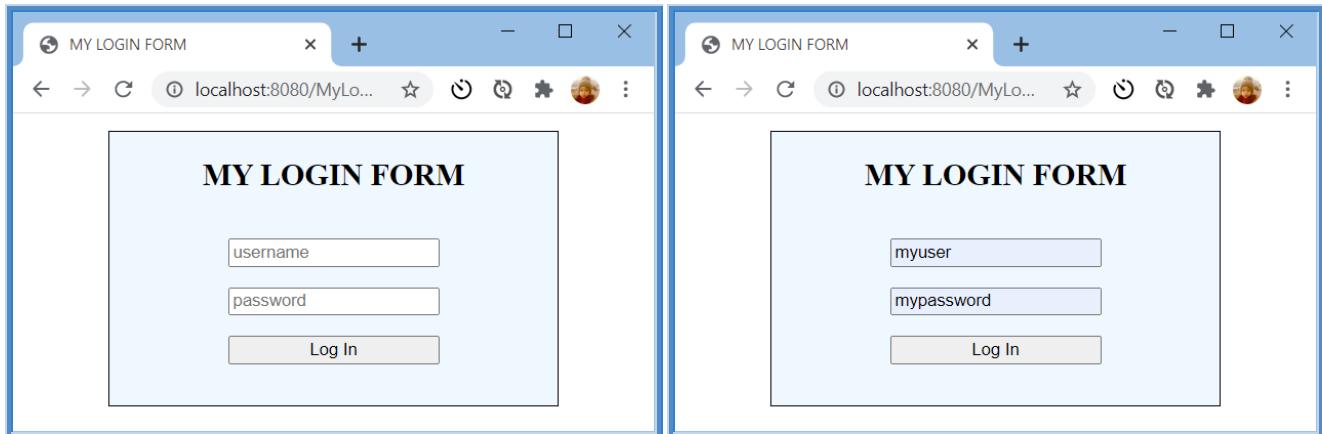
<style type="text/css">
    div { display:flex; flex-direction:column; align-items:center; border: solid 1pt; margin: 10pt 50pt;
background-color: aliceblue }
</style>

<div>
    <h2> MY LOGIN </h2>
    <form method="POST" action="/login">
        <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>
```

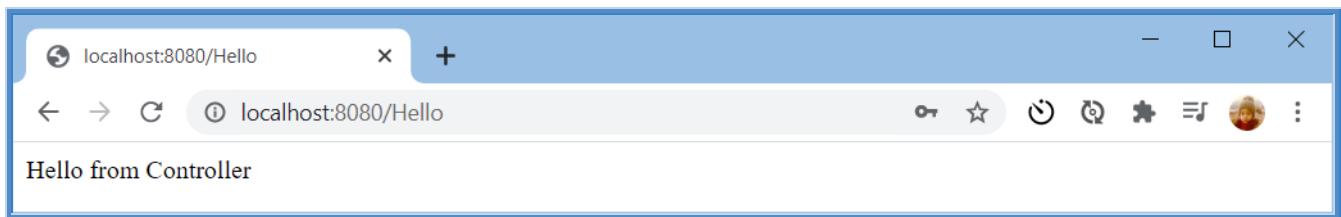
Results

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

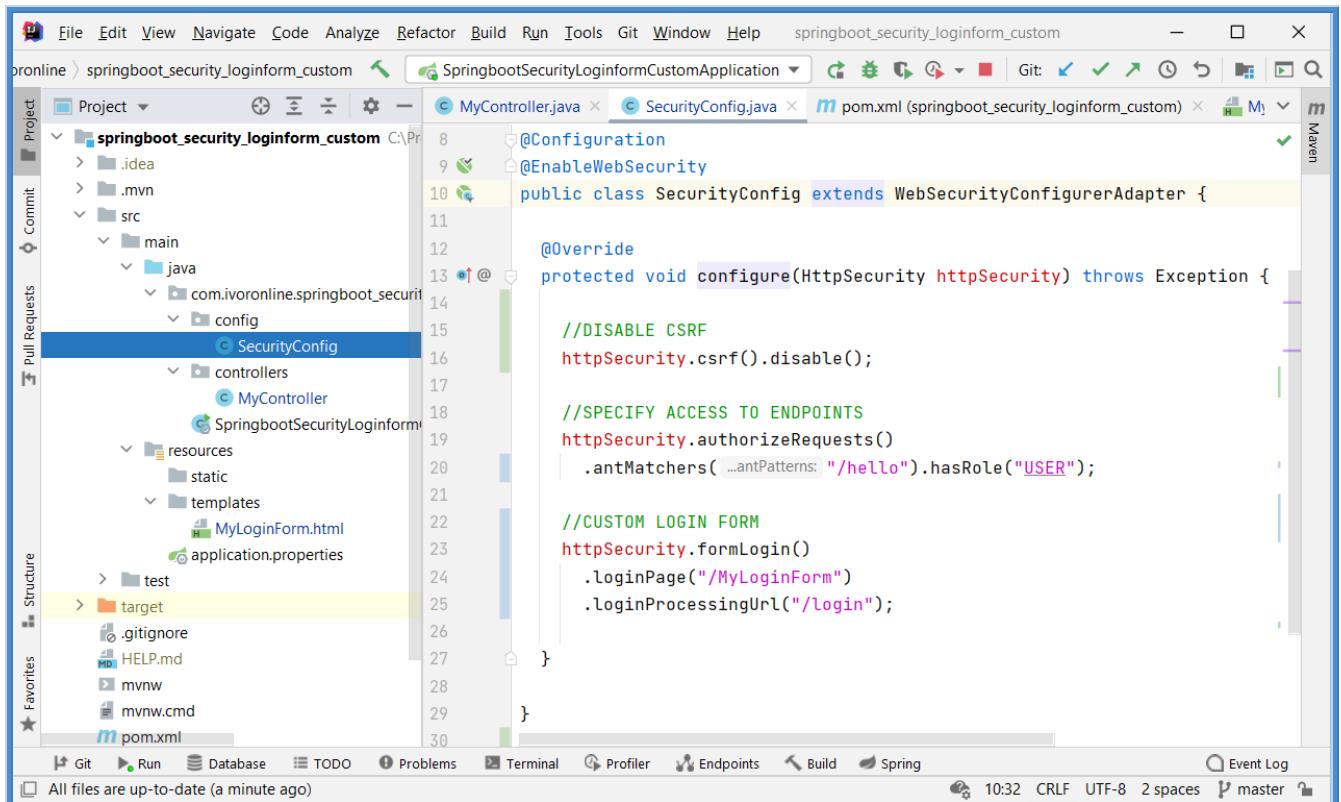
<http://localhost:8080/MyLogin>



Redirects to <http://localhost:8080>Hello>



Application Structure



```
<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>
```

1.3.3 Automatic - Authorization Header - Postman - Add

Info

- This tutorial shows how to use Postman to **add** Base64 encoded Authorization Header with Username & Password.
 - Encode Username & Password (using Online Service)
 - Add Authorization Header (in Postman using generated string from previous step)
- Postman will send HTTP Request to Spring Boot Application created in [Login Form - Default](#) tutorial (so you should start it)

Encode Username & Password

- <https://www.base64encode.org>
- Encode to Base64 format: **user:d255f064-6a4b-49bd-af08-766259e7895c**
- Encode **dXNlcjkMjU1ZjA2NC02YTRiLTQ5YmQtYWYwOC03NjYyNTIINzg5NWM=**

The screenshot shows a web browser window titled "Base64 Encode and Decode - On". The address bar shows "base64encode.org". The main content area has a green header with "BASE64" and tabs for "Decode" and "Encode". Below the header, a message says "Have to deal with Base64 format? Then this site is made for you! Use our super handy online tool to decode or encode your data." A section titled "Encode to Base64 format" contains instructions: "Simply enter your data then push the encode button." A text input field contains "user:d255f064-6a4b-49bd-af08-766259e7895c". A large green button labeled "ENCODE" is shown, with a tooltip "Encodes your data into the textarea below." To the right of the button, the encoded output "dXNlcjkMjU1ZjA2NC02YTRiLTQ5YmQtYWYwOC03NjYyNTIINzg5NWM=" is displayed in a text area.

Add Authorization Header

- Start [Login Form - Default](#) (Postman will send HTTP Request to Spring Boot Application created in this tutorial)
- Start Postman
- GET: <http://localhost:8080>Hello>
- Headers: (copy from below)
- Send

Headers

(add Key-Value)

Authorization : **dXNlcjkMjU1ZjA2NC02YTRiLTQ5YmQtYWYwOC03NjYyNTI1Nzg5NWM=**

Add Authorization Header

The screenshot shows the Postman application interface. At the top, there's a navigation bar with 'File', 'Edit', 'View', 'Help', and various tool buttons. Below the navigation is a toolbar with 'New', 'Import', 'Runner', and other options. The main workspace is titled 'My Workspace' and contains three requests: 'addAuthorBook' (GET), 'MyRequest' (POST), and another 'addAuthorBook' (POST). A sidebar on the left shows a tree view of the workspace. On the right, there are sections for 'Examples', 'BUILD', and 'Send/Save'. The bottom section shows the request details for 'addAuthorBook'.

Request Details:

Method: GET
URL: http://localhost:8080/hello

Headers Tab:

The 'Headers' tab is selected, showing a table of auto-generated headers:

KEY	VALUE	DESCRIPTION	...	Bulk Edit	Presets
Postman-Token	<calculated when request is sent>				
Host	<calculated when request is sent>				
User-Agent	PostmanRuntime/7.26.8				
Accept	*/*				
Accept-Encoding	gzip, deflate, br				
Connection	keep-alive				
Authorization	Basic dXNlcjkMjU1ZjA2NC02YTRiLTQ5YmQtYWYwOC03NjYyNTI1Nzg5NWM=				

Body Tab:

Pretty, Raw, Preview, Visualize, Text, JSON

Test Results:

Status: 200 OK, Time: 252 ms, Size: 444 B, Save Response

Bottom Navigation:

Find and Replace, Console, Bootcamp, Build, Browse

1.3.4 Automatic - Authorization Header - Postman - Generate

Info

- This tutorial shows how to use Postman to **generate** Base64 encoded Authorization Header with Username & Password.
- Encoded string represents `user:d255f064-6a4b-49bd-af08-766259e7895c`.
- Postman will send HTTP Request to Spring Boot Application created in [Login Form - Default](#) tutorial (so you should start it)

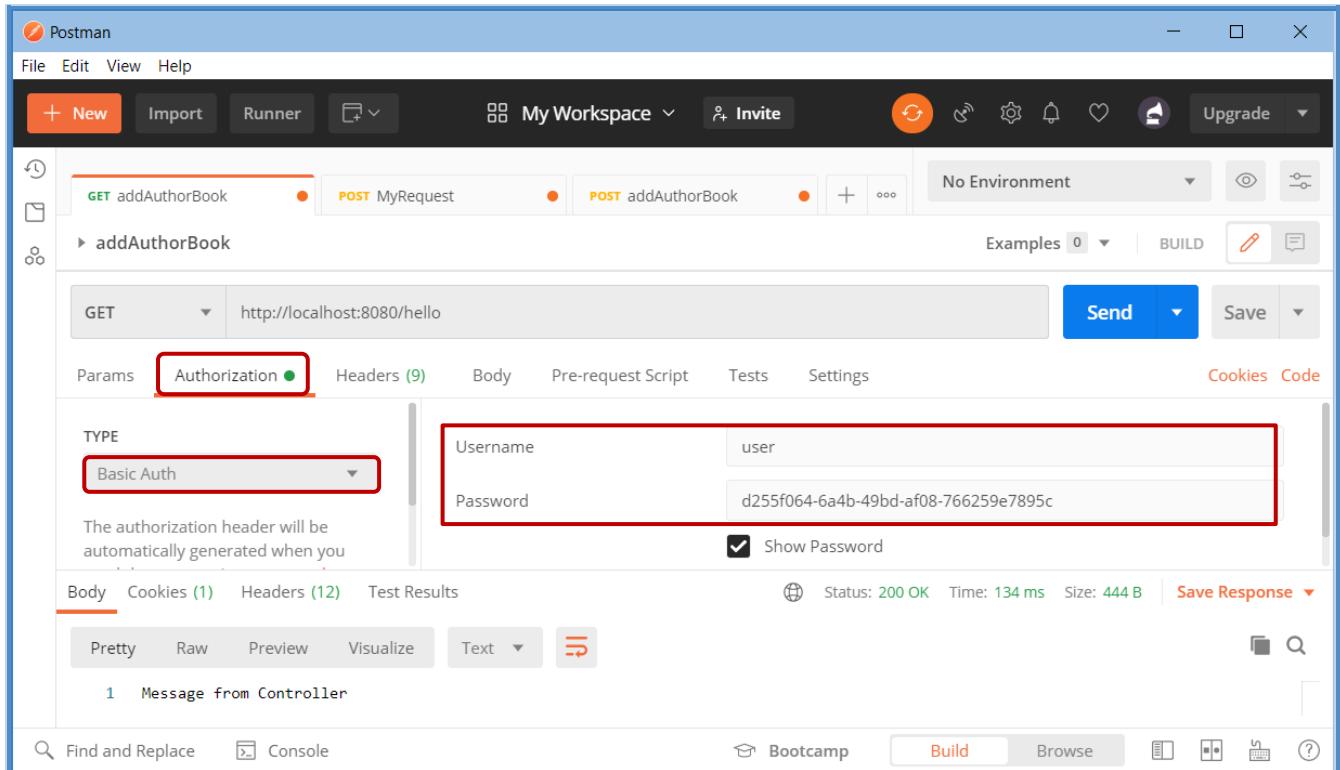
Authorization

```
Basic dXNlcjkMjU1ZjA2NC02YTRiLTQ5YmQtYWYwOC03NjYyNT11Nzg5NW
```

Procedure

- Start [Login Form - Default](#) (Postman will send HTTP Request to Spring Boot Application created in this tutorial)
- Start Postman
- GET: <http://localhost:8080/hello>
- Authorization
 - Type: Basic Auth
 - Username: user
 - Password: d255f064-6a4b-49bd-af08-766259e7895c (copy from the Console when Application starts)
- Send

Authorization - Basic Auth



Generated Authorization Header

The screenshot shows the Postman application interface. At the top, there's a navigation bar with 'File', 'Edit', 'View', 'Help', 'My Workspace' (with an 'Invite' button), and various icons for refresh, search, and notifications. Below the navigation is a toolbar with buttons for '+ New', 'Import', 'Runner', and 'Invite'. The main workspace shows a collection named 'addAuthorBook' containing three items: 'GET addAuthorBook', 'POST MyRequest', and 'POST addAuthorBook'. A status bar at the bottom indicates 'No Environment' and 'Examples 0'. On the right side, there are buttons for 'BUILD', 'Send', and 'Save'. The central part of the screen displays a request configuration for a 'GET' method to 'http://localhost:8080/hello'. Below this, the 'Headers' tab is selected, showing two entries: 'Authorization' with the value 'Basic dXNlcjkMjU1ZjA2NC02YTRiLTQ5YmQtYWYwOC03NjYyNTIINzg5NWM=' and 'Cookie' with the value 'JSESSIONID=8C0ADADC04759345C94EEB146292BB3F'. The 'Headers' tab has a red border around it.

1.3.5 Validate Credentials - Request - GET

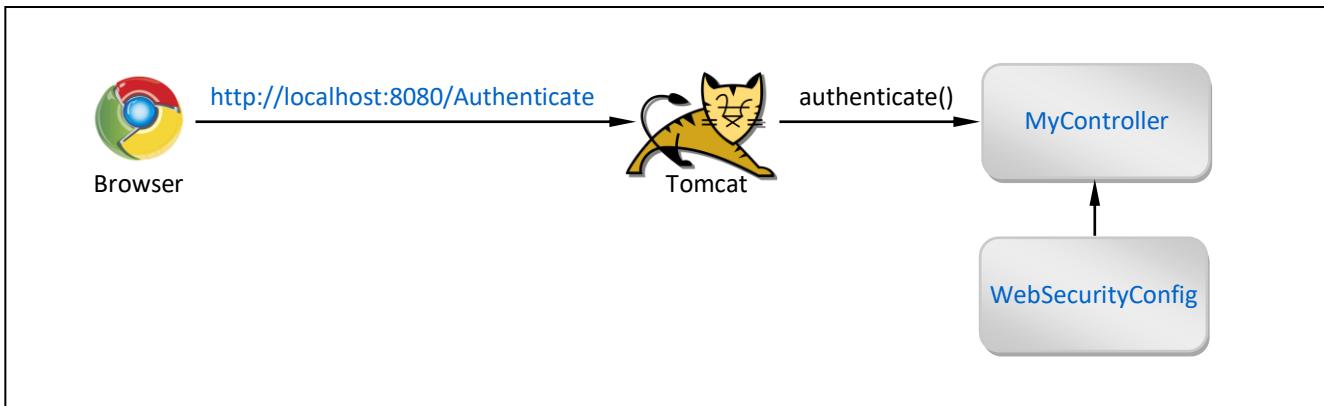
Info

[G]

- This tutorial shows how to manually Authenticate User by providing User Credentials in HTTP Request Parameters. Then we will compare provided User Credentials with stored Users on order to Authenticate User.
- Authentication will only check if User exist but it will not Authenticate User in a sense to give him access to Endpoints. User Object that is returned does not contain Property which says that User is Authenticated. Instead Authenticated Users need to be placed in a special pool to allow them access to restricted Endpoint. And we will not be doing that in this tutorial.

Application Schema

[Results]



Spring Boot Starters

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

Procedure

- **Create Project:** `springboot_security_request_parameters` (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)

WebSecurityConfig.java

```
package com.ivoronline.springboot_security_request_parameters.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.EnableWebSecurity;
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.provisioning.InMemoryUserDetailsManager;

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

    //=====
    // USER DETAILS SERVICE
    //=====

    @Bean
    @Override
    protected UserDetailsService userDetailsService() {

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

        return new InMemoryUserDetailsManager(user);
    }

    //=====
    // CONFIGURE
    //=====

    @Override
    protected void configure(HttpSecurity httpSecurity) throws Exception {
        httpSecurity.authorizeRequests().antMatchers("/Authenticate").permitAll(); //ANONYMOUS ACCESS (NO LOGIN)
    }

    //=====
    // AUTHENTICATION MANAGER BEAN
    //=====

    @Bean
    @Override
    public AuthenticationManager authenticationManager() throws Exception {
        return super.authenticationManager();
    }
}
```

MyController.java

```
package com.ivorononline.springboot_security_request_parameters.controllers;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.authentication.AuthenticationManager;
import org.springframework.security.authentication.BadCredentialsException;
import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UserDetailsService;
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 {

    @Autowired AuthenticationManager authenticationManager;
    @Autowired private UserDetailsService userDetailsService;

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

        //CREATE TOKEN (FROM USERNAME & PASSWORD)
        UsernamePasswordAuthenticationToken authToken = new UsernamePasswordAuthenticationToken(username,
password);

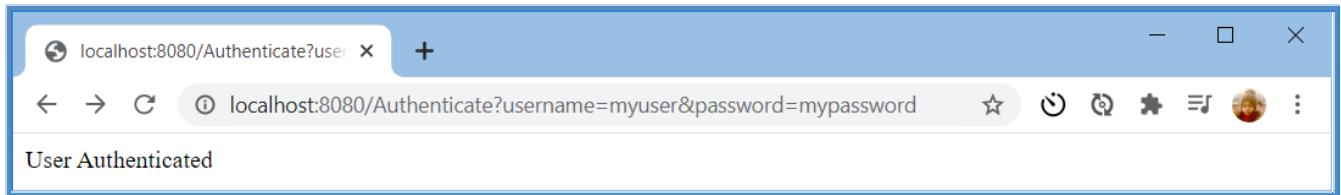
        //AUTHENTICATE
        try {
            authenticationManager.authenticate(authToken);
        } catch (BadCredentialsException e) {
            return "Invalid Credentials";
        }

        //GET USER OBJECT
        UserDetails userDetails = userDetailsService.loadUserByUsername(username);
        System.out.println(userDetails);

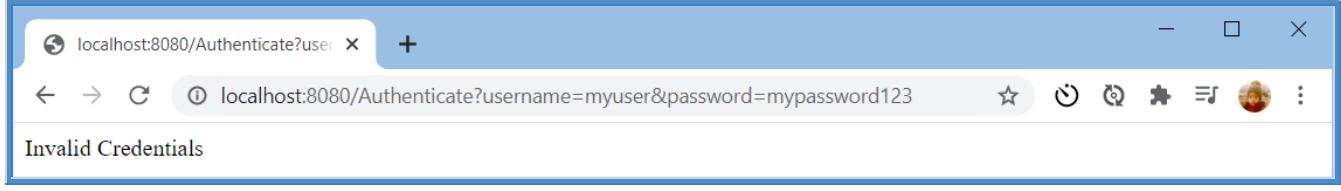
        //SUCCESSFUL AUTHENTICATION
        return "Valid Credentials";
    }
}
```

Results

<http://localhost:8080/Authenticate?username=myuser&password=mypassword>



<http://localhost:8080/Authenticate?username=myuser&password=mypassword123>



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>
```

1.3.6 Validate Credentials - Request - POST

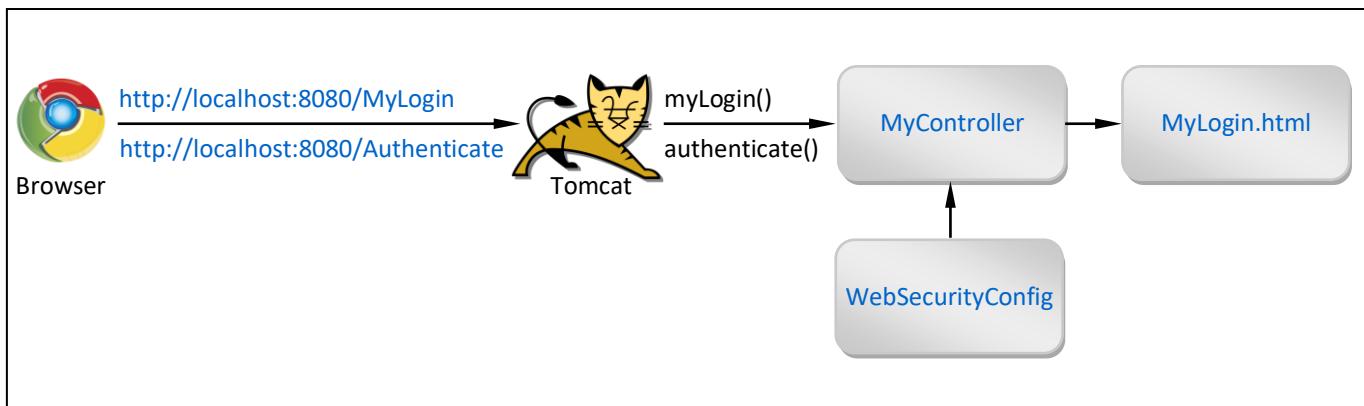
Info

[G]

- This tutorial shows how to manually Authenticate User by providing User Credentials in HTTP Request Parameters. Then we will compare provided User Credentials with stored Users on order to Authenticate User.
- Authentication will only check if User exist but it will not Authenticate User in a sense to give him access to Endpoints. User Object that is returned does not contain Property which says that User is Authenticated. Instead Authenticated Users need to be placed in a special pool to allow them access to restricted Endpoint. And we will not be doing that in this tutorial.

Application Schema

[Results]



Spring Boot Starters

GROUP	DEPENDENCY	DESCRIPTION
Web	Spring Web	Enables @RequestMapping and Tomcat
Security	Spring Security	Enables Spring Security
Template Engines	Thymeleaf	Enables Controller to return reference to HTML Page MyLogin.html

Procedure

- Create Project: `springboot_security_request_post` (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)
- Create HTML File: `MyLogin.html` (inside directory resources/templates)

WebSecurityConfig.java

```
package com.ivoronline.springboot_security_request_post.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.EnableWebSecurity;
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.provisioning.InMemoryUserDetailsManager;

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

    //=====
    // USER DETAILS SERVICE
    //=====

    @Bean
    @Override
    protected UserDetailsService userDetailsService() {

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

        return new InMemoryUserDetailsManager(user);

    }

    //=====
    // CONFIGURE
    //=====

    @Override
    protected void configure(HttpSecurity httpSecurity) throws Exception {
        httpSecurity.authorizeRequests().antMatchers("/MyLogin", "/Authenticate").permitAll(); //ANONYMOUS ACCESS
        httpSecurity.csrf().disable(); //Otherwise POST to Authenticate fails
    }

    //=====
    // AUTHENTICATION MANAGER BEAN
    //=====

    @Bean
    @Override
    public AuthenticationManager authenticationManagerBean() throws Exception {
        return super.authenticationManagerBean();
    }

}
```

MyController.java

```
package com.ivorononline.springboot_security_request_parameters.controllers;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.authentication.AuthenticationManager;
import org.springframework.security.authentication.BadCredentialsException;
import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UserDetailsService;
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 {

    @Autowired AuthenticationManager authenticationManager;
    @Autowired private UserDetailsService userDetailsService;

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

        //CREATE TOKEN (FROM USERNAME & PASSWORD)
        UsernamePasswordAuthenticationToken authToken = new UsernamePasswordAuthenticationToken(username,
password);

        //AUTHENTICATE
        try {
            authenticationManager.authenticate(authToken);
        } catch (BadCredentialsException e) {
            return "Invalid Credentials";
        }

        //GET USER OBJECT
        UserDetails userDetails = userDetailsService.loadUserByUsername(username);
        System.out.println(userDetails);

        //SUCCESSFUL AUTHENTICATION
        return "Valid Credentials";
    }
}
```

MyLogin.html

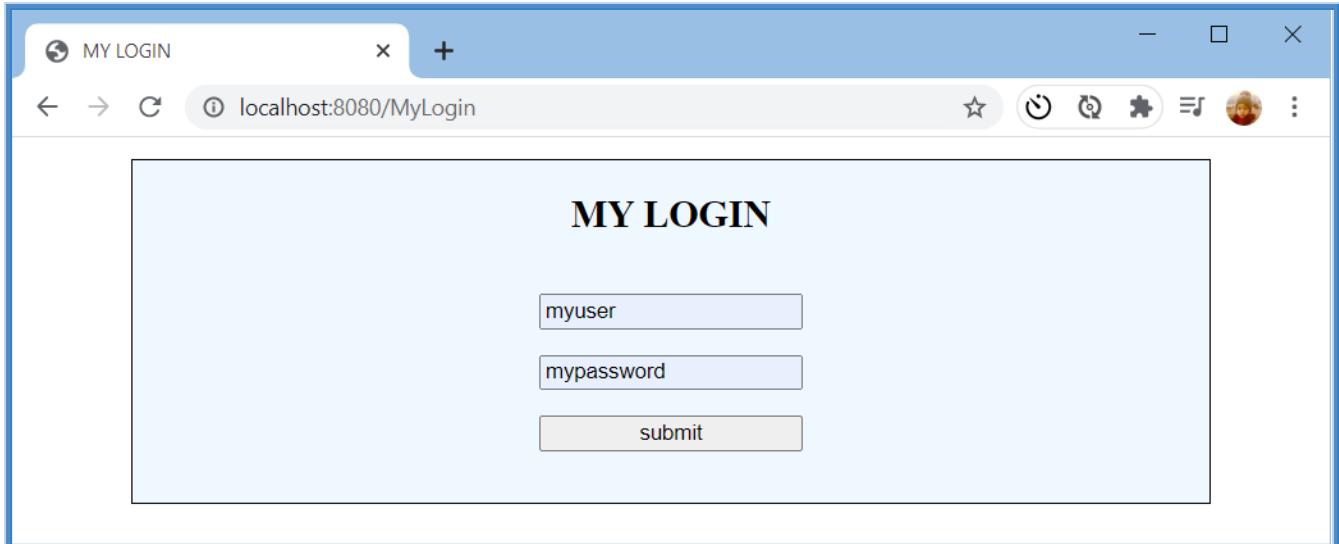
```
<title> MY LOGIN </title>

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

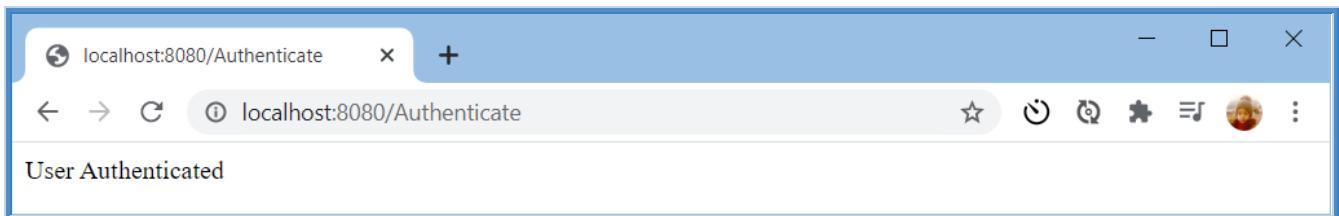
<div>
    <h2> MY LOGIN </h2>
    <form method="POST" action="/Authenticate">
        <p> <input type="text" name="username" placeholder="username" /> </p>
        <p> <input type="text" name="password" placeholder="password" /> </p>
        <p> <input type="submit" name="submit" value="submit" style="width:100%"/>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

Results

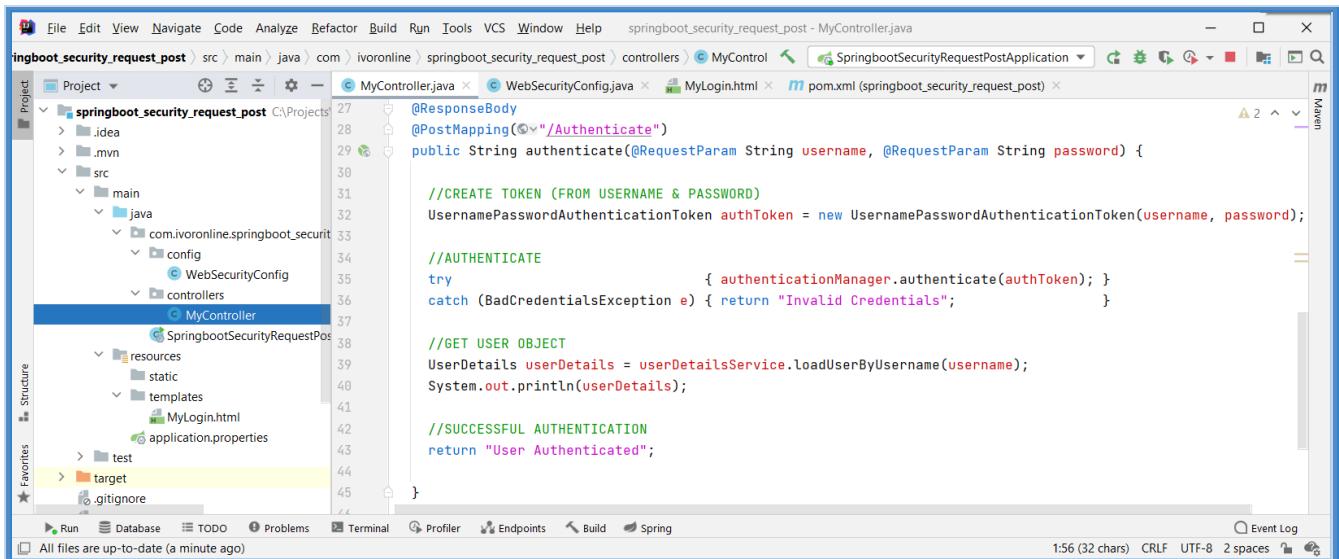
<http://localhost:8080/MyLogin>



<http://localhost:8080/Authenticate>



Application Structure



```
<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>
```

1.3.7 Validate Credentials - Request - POST - JSON

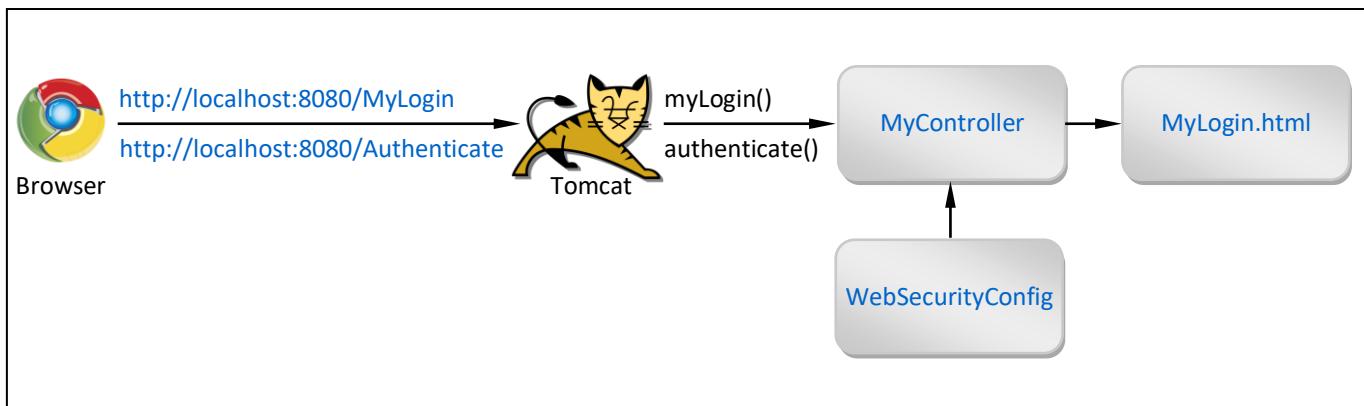
Info

[G]

- This tutorial shows how to manually Authenticate User by providing User Credentials in HTTP Request Parameters. Then we will compare provided User Credentials with stored Users on order to Authenticate User.
- Authentication will only check if User exist but it will not Authenticate User in a sense to give him access to Endpoints. User Object that is returned does not contain Property which says that User is Authenticated. Instead Authenticated Users need to be placed in a special pool to allow them access to restricted Endpoint. And we will not be doing that in this tutorial.

Application Schema

[Results]



Spring Boot Starters

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

Procedure

- Create Project:** `springboot_security_request_post` (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)

WebSecurityConfig.java

```
package com.ivoronline.springboot_security_request_post.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.EnableWebSecurity;
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.provisioning.InMemoryUserDetailsManager;

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

    //=====
    // USER DETAILS SERVICE
    //=====

    @Bean
    @Override
    protected UserDetailsService userDetailsService() {

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

        return new InMemoryUserDetailsManager(user);
    }

    //=====
    // CONFIGURE
    //=====

    @Override
    protected void configure(HttpSecurity httpSecurity) throws Exception {
        httpSecurity.authorizeRequests().antMatchers("/Authenticate").permitAll(); //ANONYMOUS ACCESS
        httpSecurity.csrf().disable(); //Otherwise POST to Authenticate fails
    }

    //=====
    // AUTHENTICATION MANAGER BEAN
    //=====

    @Bean
    @Override
    public AuthenticationManager authenticationManager() throws Exception {
        return super.authenticationManager();
    }
}
```

MyController.java

```
package com.ivorononline.springboot_security_request_post_json.controllers;

import com.ivorononline.springboot_security_request_post_json.DTO.CredentialsDTO;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.authentication.AuthenticationManager;
import org.springframework.security.authentication.BadCredentialsException;
import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.web.bind.annotation.RequestBody;
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 {

    @Autowired AuthenticationManager authenticationManager;
    @Autowired private UserDetailsService userDetailsService;

    @ResponseBody
    @RequestMapping("/Authenticate")
    public String authenticate(@RequestBody CredentialsDTO credentialsDTO) {

        //CREATE TOKEN (FROM USERNAME & PASSWORD)
        String username = credentialsDTO.getUsername();
        String password = credentialsDTO.getPassword();
        UsernamePasswordAuthenticationToken authToken = new UsernamePasswordAuthenticationToken(username,
password);

        //AUTHENTICATE
        try {
            authenticationManager.authenticate(authToken);
        } catch (BadCredentialsException e) {
            return "Invalid Credentials";
        }

        //GET USER OBJECT
        UserDetails userDetails = userDetailsService.loadUserByUsername(username);
        System.out.println(userDetails);

        //SUCCESSFUL AUTHENTICATION
        return "Valid Credentials";
    }
}
```

Results

- Start Postman
- POST: <http://localhost:8080/Authenticate>
- Headers: (paste from below)
- Body: (paste from below)
- Send

Headers

(add Key-Value)

```
Content-Type: application/json
```

Body

(option: raw)

```
{
  "username" : "myuser",
  "password" : "mypassword"
}
```

Postman

The screenshot shows the Postman application window. At the top, there's a navigation bar with File, Edit, View, Help, Home, Workspaces, Reports, Explore, and a search bar. Below the navigation is a toolbar with various icons. The main area shows a list of requests under 'Test2 / JWTPassword'. A specific POST request to 'http://localhost:8080/Authenticate' is selected. The 'Body' tab is active, showing a JSON payload:

```
{
  "username" : "myuser",
  "password" : "mypassword"
}
```

Below the body, the status bar indicates 'Status: 200 OK Time: 105 ms Size: 364 B'. The bottom of the window has tabs for Body, Cookies, Headers, Test Results, and a 'Pretty' button.

Application Structure

The screenshot shows the IntelliJ IDEA IDE interface. The top menu includes File, Edit, View, Navigate, Code, Analyze, Refactor, Build, Run, Tools, VCS, Window, Help. The code editor displays a Java file named 'MyController.java' with the following code:

```
@RequestMapping(value = "/Authenticate")
public String authenticate(@RequestBody CredentialsDTO credentialsDTO) {
    //CREATE TOKEN (FROM USERNAME & PASSWORD)
    String username = credentialsDTO.getUsername();
    String password = credentialsDTO.getPassword();
    UsernamePasswordAuthenticationToken authToken = new UsernamePasswordAuthenticationToken(username, password);

    //AUTHENTICATE
    try {
        authenticationManager.authenticate(authToken);
    } catch (BadCredentialsException e) {
        return "Invalid Credentials";
    }

    //GET USER OBJECT
    UserDetails userDetails = userDetailsService.loadUserByUsername(username);
    System.out.println(userDetails);

    //SUCCESSFUL AUTHENTICATION
    return "User Authenticated";
}
```

The left side of the screen shows the project structure with packages like 'com.ivoronline.springboot_security' and files like 'WebSecurityConfig.java' and 'CredentialsDTO.java'. The bottom status bar shows 'All files are up-to-date (7 minutes ago)' and other system information.

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>
```

1.3.8 Manually Create User Object

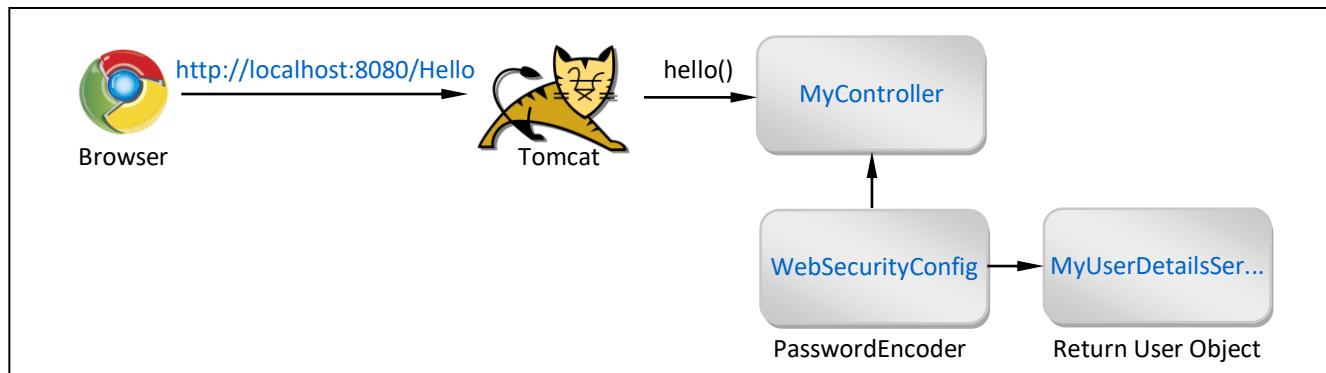
Info

[G]

- This tutorial shows how to manually create User Object inside [MyUserDetailsService.java](#).
 - User is created and returned only if entered username matches stored username (otherwise Exception is thrown).
 - In this simplified example we only have one stored User against which entered Username is checked.
- Tutorial [Define Users in DB](#) shows how to compare entered Username against Users stored in Database.

Application Schema

[Results]



Spring Boot Starters

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

MyUserDetailsService.java

- If you
 - declare Class that `implements UserDetailsService` (it is called Service because it SERVES UserDetails Object)
 - and `@Override` its Method `loadUserByUsername(String username)`
 - then this Method will be called when User enters Username and Password through Login Form
- Method
 - should return Object of Class that Implements `UserDetails` Interface (which in our case will be built-in User Class)
 - which should contain: Username, **Password**, Authorities
- This User Object
 - will contain **Stored Password**
 - which Spring will compare with the **Entered Password** (from Login Form)

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: `config` (inside main package)
 - Create Class: `WebSecurityConfig.java` (inside package config)
- Create Package: `services` (inside main package)
 - Create Class: `MyUserDetailsService.java` (inside package services)

`MyUserDetailsService.java`

```
package com.ivorononline.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.List;

@Service
public class MyUserDetailsService implements UserDetailsService {

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

        //HARD CODED USER
        String username = "myuser";
        String password = "mypassword";
        String role     = "ROLE_USER";

        //CHECK USERNAME
        if (!username.equals(enteredUsername)) {
            throw new UsernameNotFoundException(enteredUsername);
        }

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

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

        //RETURN USER
        return user;
    }
}
```

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

    @Bean
    PasswordEncoder passwordEncoder() {
        return NoOpPasswordEncoder.getInstance();
    }

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

        //RESTRICT ACCESS TO EVERYTHING ELSE (BEHIND LOGIN FORM)
        httpSecurity
            .authorizeRequests().anyRequest().authenticated()
            .and().formLogin()
            .and().httpBasic();

    }
}
```

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

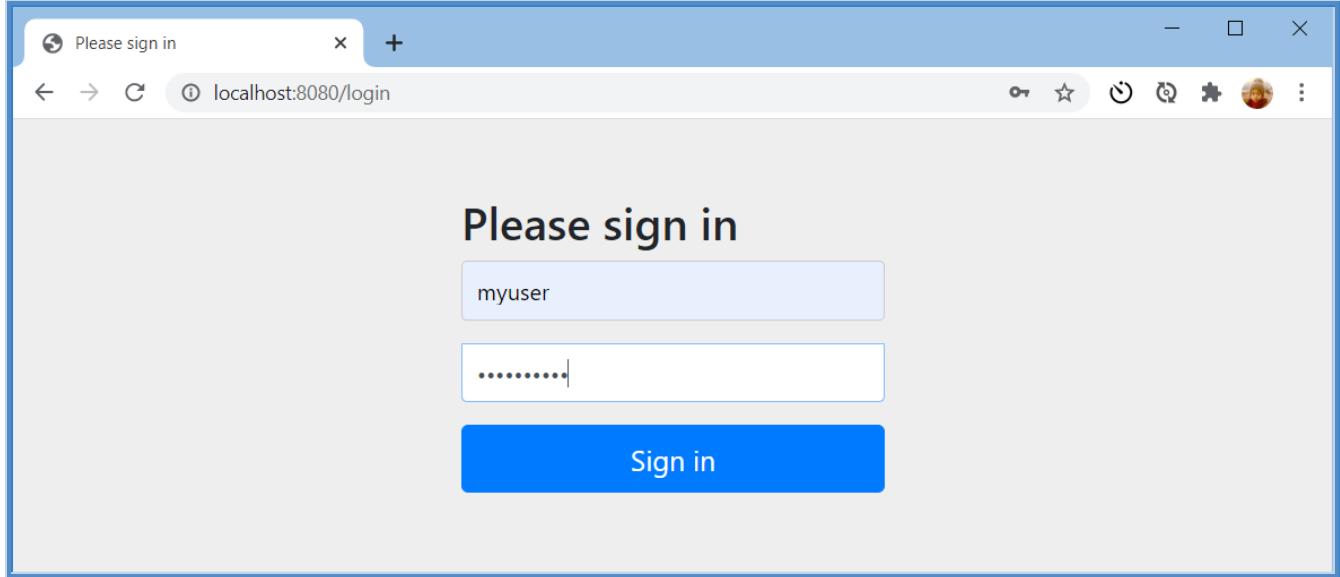
@Controller
public class MyController {

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

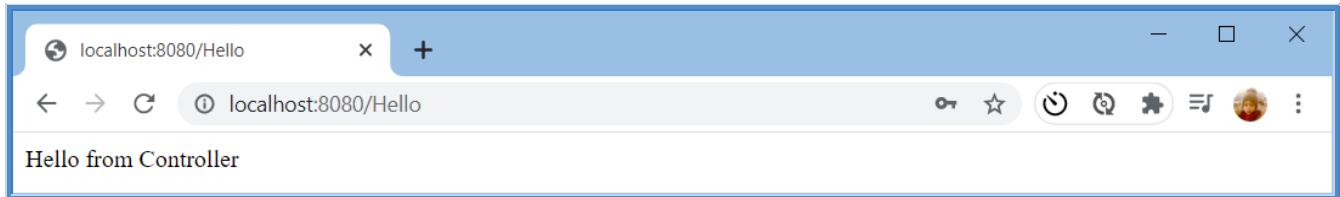
Results

- <http://localhost:8080>Hello>
- You get redirected to <http://localhost:8080/login>
 - Username: **john**
 - Password: **johnpassword**
 - Sign in
- You get redirected back to <http://localhost:8080>Hello>

<http://localhost:8080/login> - john - johnpassword



Redirects to <http://localhost:8080>Hello>



Application Structure

The screenshot shows the IntelliJ IDEA interface with the following details:

- Project View:** Shows the project structure under "springboot_manuallycreateuserobject". The "src/main/java" folder contains packages like "com.ivoronline.springboot_manual" which include "config", "controllers", and "services". The "services" package contains a class named "MyUserDetailsService".
- Code Editor:** The code for "MyUserDetailsService.java" is displayed. It implements the "UserDetailsService" interface. The implementation involves hard-coding user details, checking the entered username, creating authorities, and finally creating and returning a User object.
- Toolbars and Menus:** Standard IntelliJ menus like File, Edit, View, Navigate, Code, Analyze, Refactor, Build, Run, Tools, VCS, Window, Help are visible at the top.
- Bottom Status Bar:** Shows "Build completed successfully in 1 sec, 901 ms (a minute ago)" and other build-related information.

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>
```

1.3.9 Events - Log to Console

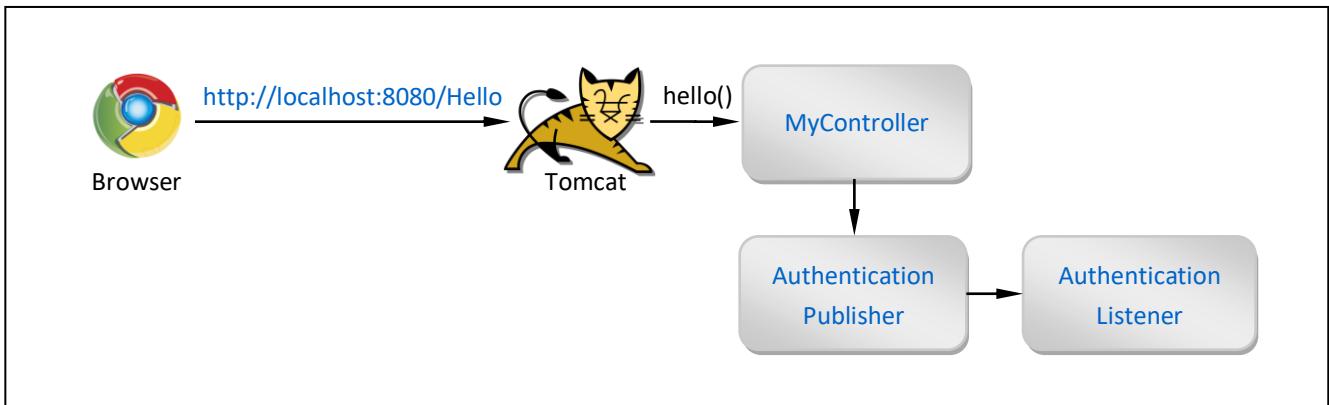
Info

[G]

- This tutorial shows how to Publish and Listen for Authentication Events (Success and Failure).
- These Events are triggered after User enters Username and Password (valid or invalid).

Application Schema

[Results]



Spring Boot Starters

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

Procedure

- **Create Project:** `springboot_security_events` (add Spring Boot Starters from the table)
- **Edit File:** `application.properties` (add Role, User, Password)
- **Create Package:** `config` (inside main package)
 - **Create Class:** `AuthenticationPublisher.java` (inside config package)
 - **Create Class:** `AuthenticationListener.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 = mypassword
spring.security.user.roles     = USER
```

MyController.java

```
package com.ivoronline.springbott_security_events.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 hello() {
        return "Hello from Controller";
    }
}
```

AuthenticationPublisher.java

```
package com.ivoronline.springbott_security_events.config;

import org.springframework.context.ApplicationEventPublisher;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.authentication.AuthenticationEventPublisher;
import org.springframework.security.authentication.DefaultAuthenticationEventPublisher;

@Configuration
public class AuthenticationPublisher {

    @Bean
    public AuthenticationEventPublisher publish(ApplicationEventPublisher applicationEventPublisher){
        return new DefaultAuthenticationEventPublisher(applicationEventPublisher);
    }
}
```

AuthenticationListener.java

```
package com.ivorononline.springbott_security_events.config;

import org.springframework.context.event.EventListener;
import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;
import org.springframework.security.authentication.event.AuthenticationFailureBadCredentialsEvent;
import org.springframework.security.authentication.event.AuthenticationSuccessEvent;
import org.springframework.security.core.userdetails.User;
import org.springframework.security.web.authentication.WebAuthenticationDetails;
import org.springframework.stereotype.Component;

@Component
public class AuthenticationListener {

    //=====
    // METHOD: LISTEN TO SUCCESS
    //=====

    @EventListener
    public void listenToSuccess(AuthenticationSuccessEvent event){

        //GET USERNAME
        UsernamePasswordAuthenticationToken token = (UsernamePasswordAuthenticationToken) event.getSource();
        User user = (User) token.getPrincipal();
        String username = user.getUsername();

        //GET IP
        WebAuthenticationDetails details = (WebAuthenticationDetails) token.getDetails();
        String IP = details.getRemoteAddress();

        //LOG DATA
        System.out.println("Successful Login by Username/IP: " + username + "/" + IP);

    }

    //=====
    // METHOD: LISTEN TO FAILURE
    //=====

    @EventListener
    public void listenToFailure(AuthenticationFailureBadCredentialsEvent event){

        //GET USERNAME
        UsernamePasswordAuthenticationToken token = (UsernamePasswordAuthenticationToken) event.getSource();
        String username = (String) token.getPrincipal();

        //GET IP
        WebAuthenticationDetails details = (WebAuthenticationDetails) token.getDetails();
        String IP = details.getRemoteAddress();

        //LOG DATA
        System.out.println("Unsuccessful Login by Username/IP: " + username + "/" + IP);

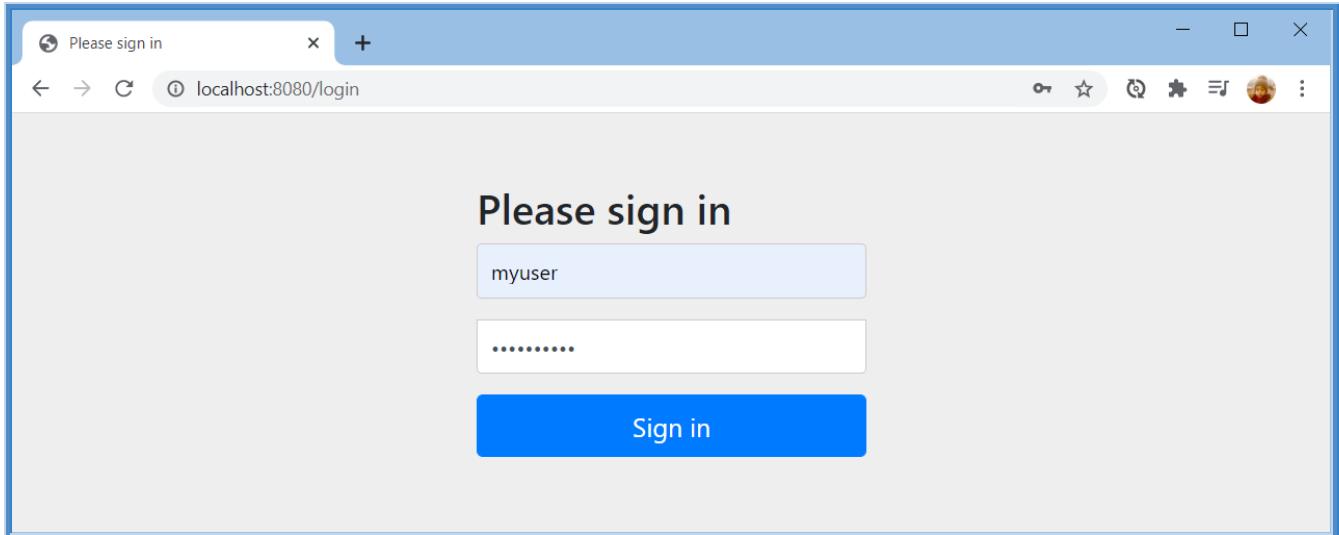
    }

}
```

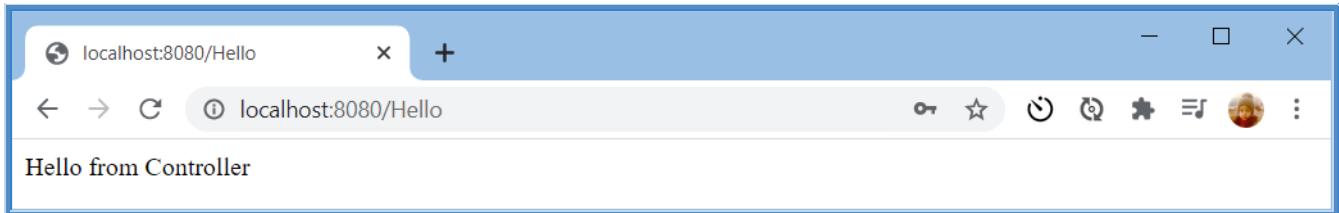
Results

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

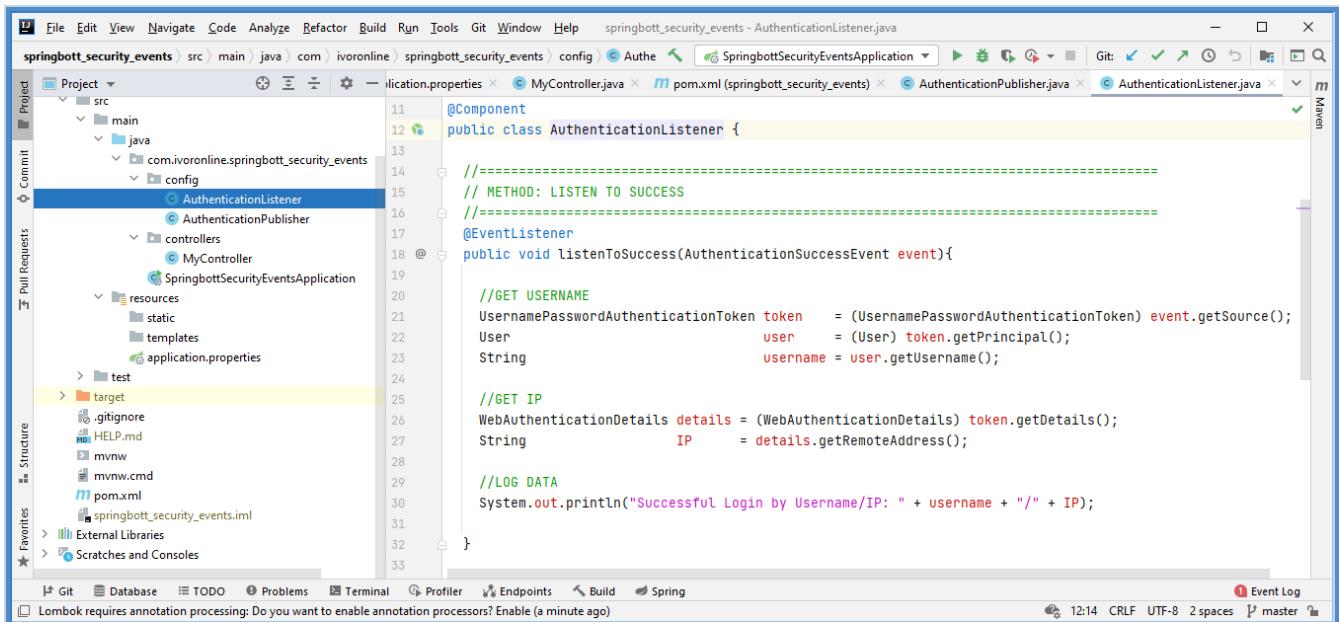
<http://localhost:8080/login>



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>
```

1.4 Authorization

Info

- Following tutorials show a different ways of specifying which Roles have access to which Endpoints.
- Since Roles are assigned to Users this defines which Users have access to which Endpoints (depending on their Roles).
- Or in other words which Roles (Users) are authorized to access which Endpoints.
- [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)
```

- [Security Expressions - @Secured](#) allows you to define access through Annotations that can only specify Roles.

SecurityConfig.java

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

MyController.java

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

- [Security Expressions - @PreAuthorize](#) allows you to define access through Annotations that can use Expressions.

SecurityConfig.java

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

MyController.java

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

- [Ant Matchers](#) are used to specify URL Patterns to which API Expression will be applied.

SecurityConfig.java

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

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.

1.4.1 Security Expressions - API - Roles & Authorities

Info

[G]

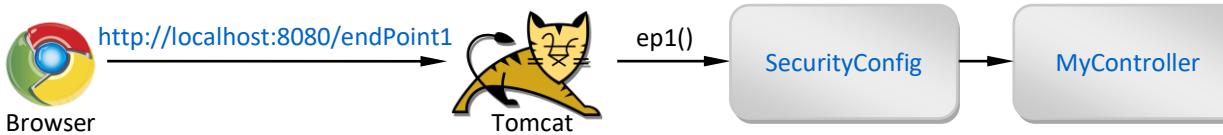
- When you add Spring Boot Starter **Security** every URL becomes unavailable unless you Sign In.
- This tutorial shows how to loosen up this restriction by allowing access to certain URLs without having to Sign In.
- We will do this by creating a Class **SecurityConfig** which extends and Overrides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

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)
```

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

application.properties

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

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("/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"; }
}
```

SecurityConfig.java

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

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;

@Configuration
@EnableWebSecurity
public class SecurityConfig extends WebSecurityConfigurerAdapter {

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

        //SPECIFY ACCESS TO ENDPOINTS
        httpSecurity.authorizeRequests()
            .antMatchers("/endPoint1").denyAll()                                //No access (even after log in)
            .antMatchers("/endPoint2").permitAll()                               //No log in (anonymous access)
            .antMatchers("/endPoint3").hasRole("ADMIN")                         //ADMIN ROLE can access AFTER log in
            .antMatchers("/endPoint4").hasAnyRole("ADMIN", "USER");           //ADMIN/USER ROLE can access AFTER log in

        //REDIRECT TO LOGIN FORM
        httpSecurity.formLogin();

    }
}
```

Results

<http://localhost:8080/endPoint1>

denyAll() => deny ALL after Login

localhost:8080/endPoint1

localhost:8080/endPoint1

Whitelabel Error Page

This application has no explicit mapping for /error, so you are seeing this as a fallback.

Tue Jan 19 16:10:31 CET 2021

There was an unexpected error (type=Forbidden, status=403).

<http://localhost:8080/endPoint2>

permitAll() => allow ALL without Login

localhost:8080/endPoint2

localhost:8080/endPoint2

endPoint2

<http://localhost:8080/endPoint3>

hasRole("ADMIN") => deny USER after Login

localhost:8080/endPoint3

localhost:8080/endPoint3

Whitelabel Error Page

This application has no explicit mapping for /error, so you are seeing this as a fallback.

Tue Jan 19 16:12:31 CET 2021

There was an unexpected error (type=Forbidden, status=403).

<http://localhost:8080/endPoint4>

hasAnyRole("ADMIN", "USER") => allow USER after Login

localhost:8080/endPoint4

localhost:8080/endPoint4

endPoint4

1.4.2 Security Expressions - @Secured - Roles

Info

[G]

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

SecurityConfig.java

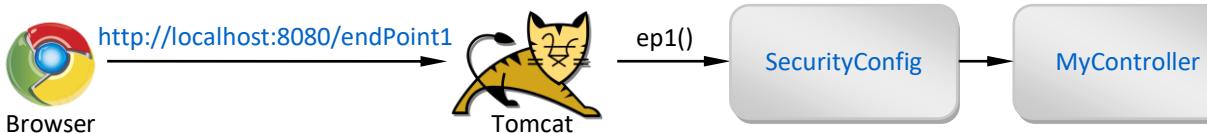
```
@EnableGlobalMethodSecurity(securedEnabled = true)
```

MyController.java

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

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

application.properties

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

SecurityConfig.java

```
package com.ivoronline.springboot_security_expression_secured.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.EnableWebSecurity;
import org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;

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

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

        //SPECIFY ACCESS TO ENDPOINTS
        httpSecurity.authorizeRequests()
            .antMatchers("/endPoint1").denyAll()      //No access (even after log in)
            .antMatchers("/endPoint2").permitAll();   //No log in (anonymous access)

        //REDIRECT TO LOGIN FORM
        httpSecurity.formLogin();

    }

}
```

MyController.java

```
package com.ivoronline.springboot_security_expression_secured.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 {

    @ResponseBody
    @RequestMapping("/endPoint1")
    public String ep1() { return "endPoint1"; }

    @ResponseBody
    @RequestMapping("/endPoint2")
    public String ep2() { return "endPoint2"; }

    @Secured("ROLE_ADMIN")
    @ResponseBody
    @RequestMapping("/endPoint3")
    public String ep3() { return "endPoint3"; }

    @Secured({"ROLE_ADMIN", "ROLE_USER"})
    @ResponseBody
    @RequestMapping("/endPoint4")
    public String ep4() { return "endPoint4"; }

}
```

Results

<http://localhost:8080/endPoint1>

denyAll() => deny ALL after Login

localhost:8080/endPoint1

Whitelabel Error Page

This application has no explicit mapping for /error, so you are seeing this as a fallback.

Tue Jan 19 16:10:31 CET 2021

There was an unexpected error (type=Forbidden, status=403).

<http://localhost:8080/endPoint2>

permitAll() => allow ALL without Login

localhost:8080/endPoint2

endPoint2

<http://localhost:8080/endPoint3>

hasRole("ADMIN") => deny USER after Login

localhost:8080/endPoint3

Whitelabel Error Page

This application has no explicit mapping for /error, so you are seeing this as a fallback.

Tue Jan 19 16:12:31 CET 2021

There was an unexpected error (type=Forbidden, status=403).

<http://localhost:8080/endPoint4>

hasAnyRole("ADMIN", "USER") => allow USER after Login

localhost:8080/endPoint4

endPoint4

1.4.3 Security Expressions - @PreAuthorize - Roles & Authorities

Info

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

SecurityConfig.java

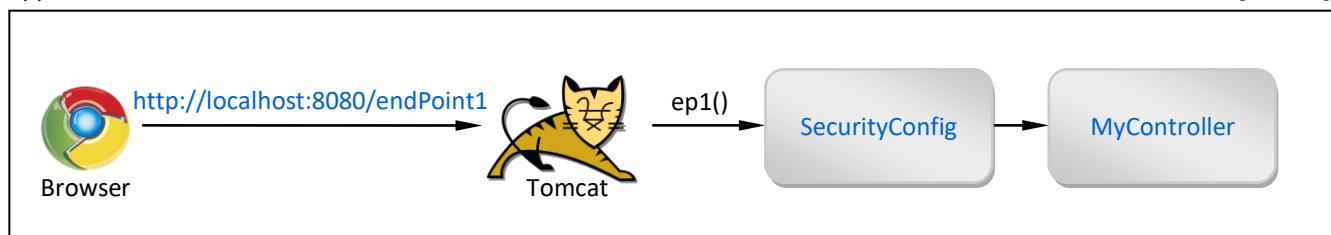
```
@EnableGlobalMethodSecurity(prePostEnabled = true)
```

MyController.java

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

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

application.properties

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

SecurityConfig.java

```
package com.ivoronline.springboot_security_expression_prauthorized.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.EnableWebSecurity;
import org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;

@Configuration
@EnableWebSecurity
@EnableGlobalMethodSecurity(prePostEnabled = true)
public class SecurityConfig extends WebSecurityConfigurerAdapter {

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

        //SPECIFY ACCESS TO ENDPOINTS
        httpSecurity.authorizeRequests()
            .antMatchers("/endPoint1").denyAll()      //No access (even after log in)
            .antMatchers("/endPoint2").permitAll();   //No log in (anonymous access)

        //REDIRECT TO LOGIN FORM
        httpSecurity.formLogin();

    }

}
```

MyController.java

```
package com.ivoronline.springboot_security_expression_prauthorized.controllers;

import org.springframework.security.access.prepost.PreAuthorize;
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("/endPoint1")
    public String ep1() { return "endPoint1"; }

    @ResponseBody
    @RequestMapping("/endPoint2")
    public String ep2() { return "endPoint2"; }

    @PreAuthorize("hasRole('ADMIN')")
    @ResponseBody
    @RequestMapping("/endPoint3")
    public String ep3() { return "endPoint3"; }

    @PreAuthorize("hasAnyRole('ADMIN', 'USER')")
    @ResponseBody
    @RequestMapping("/endPoint4")
    public String ep4() { return "endPoint4"; }

}
```

Results

<http://localhost:8080/endPoint1>

denyAll() => deny ALL after Login

localhost:8080/endPoint1

localhost:8080/endPoint1

Whitelabel Error Page

This application has no explicit mapping for /error, so you are seeing this as a fallback.

Tue Jan 19 16:10:31 CET 2021

There was an unexpected error (type=Forbidden, status=403.).

<http://localhost:8080/endPoint2>

permitAll() => allow ALL without Login

localhost:8080/endPoint2

localhost:8080/endPoint2

endPoint2

<http://localhost:8080/endPoint3>

hasRole("ADMIN") => deny USER after Login

localhost:8080/endPoint3

localhost:8080/endPoint3

Whitelabel Error Page

This application has no explicit mapping for /error, so you are seeing this as a fallback.

Tue Jan 19 16:12:31 CET 2021

There was an unexpected error (type=Forbidden, status=403.).

<http://localhost:8080/endPoint4>

hasAnyRole("ADMIN", "USER") => allow USER after Login

localhost:8080/endPoint4

localhost:8080/endPoint4

endPoint4

1.4.4 Security Expressions - @PreAuthorize - Custom Methods

Info

[G]

- This tutorial shows how to use Custom Class/Methods to Authorize access to Endpoint.
This can be combined into Security Expressions together with Roles and Authorities (as shown below).
- The idea is to create Custom Class `AuthenticationService` with Methods like `authenticate()` that return Boolean.
Then to call these Methods from `@PreAuthorize()` Annotation to control access to Endpoints (as shown below).

AuthenticationService.java

```
@Component
public class AuthenticationService {

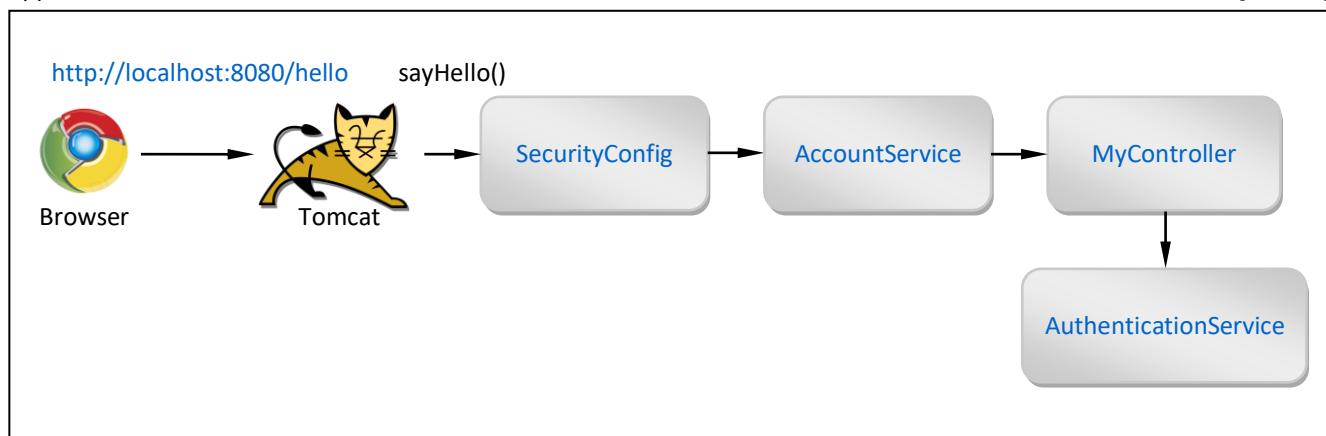
    public boolean authenticate(Authentication authentication) {
        UserDetails user = (UserDetails) authentication.getPrincipal();
        System.out.println(user.getUsername());
        return true;
    }
}
```

MyController.java

```
@PreAuthorize("hasAuthority('book.read') AND @authenticationService.authenticate(authentication)")
```

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: springboot_security_authorization_custom (add Spring Boot Starters from the table)
- Edit File: application.properties (specify Username, Password, Role)
- Create Package: controllers (inside main package)
 - Create Class: MyController.java (inside controllers package)
- Create Package: config (inside main package)
 - Create Class: SecurityConfig.java (inside config package)
- Create Package: services (inside main package)
 - Create Class: AccountService.java (inside services package)
 - Create Class: AuthenticationService.java (inside services package)

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
profile.admin = book.create, book.read, book.update, book.delete
```

MyController.java

```
package com.ivoronline.springboot_security_authorization_custom.controllers;

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

@Controller
public class MyController {

    @ResponseBody
    @PreAuthorize("hasAuthority('book.read') AND @authenticationService.authenticate(authentication)")
    @RequestMapping("/ReadBook")
    public String readBook() {
        return "ADMIN and USER can read Book";
    }

}
```

AuthenticationService.java

```
package com.ivoronline.springboot_security_authorization_custom.services;

import org.springframework.security.core.Authentication;
import org.springframework.security.core.userdetails.User;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.stereotype.Component;
import java.util.UUID;

@Component
public class AuthenticationService {

    public boolean authenticate(Authentication authentication) {
        UserDetails user = (UserDetails) authentication.getPrincipal();
        System.out.println(user.getUsername());
        return true;
    }

}
```

SecurityConfig.java

```
package com.ivoronline.springboot_security_authorization_custom.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.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
@EnableGlobalMethodSecurity(prePostEnabled = true)
public class SecurityConfig extends WebSecurityConfigurerAdapter {

    @Bean
    PasswordEncoder passwordEncoder() {
        return NoOpPasswordEncoder.getInstance();
    }

    @Override
    protected void configure(HttpSecurity httpSecurity) throws Exception {
        httpSecurity.formLogin();
    }

}
```

AccountService.java

```
package com.ivoronline.springboot_security_authorization_custom.services;

import org.springframework.beans.factory.annotation.Value;
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.Arrays;
import java.util.List;

@Service
public class AccountService 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()));
        }

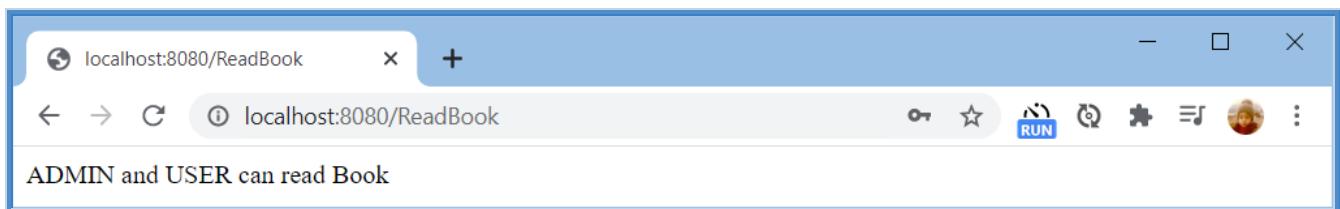
        //CREATE USER
        User user = new User("myuser", "mypassword", true, true, true, true, authorities);

        //RETURN USER
        return user;
    }
}
```

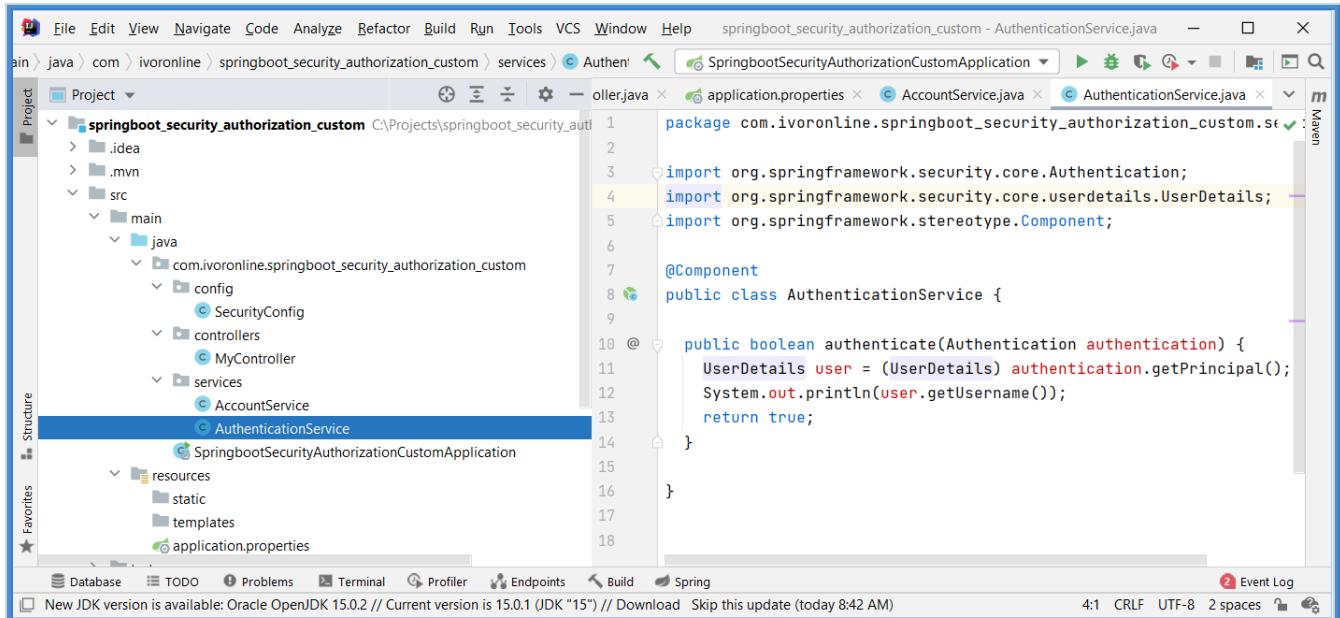
Results

<http://localhost:8080/ReadBook>

@PreAuthorize("hasAuthority('book.read') AND @authenticationService.authenticate(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>
```

1.4.5 Security Expressions - @PreAuthorize - Custom Methods - Books

Info

[G]

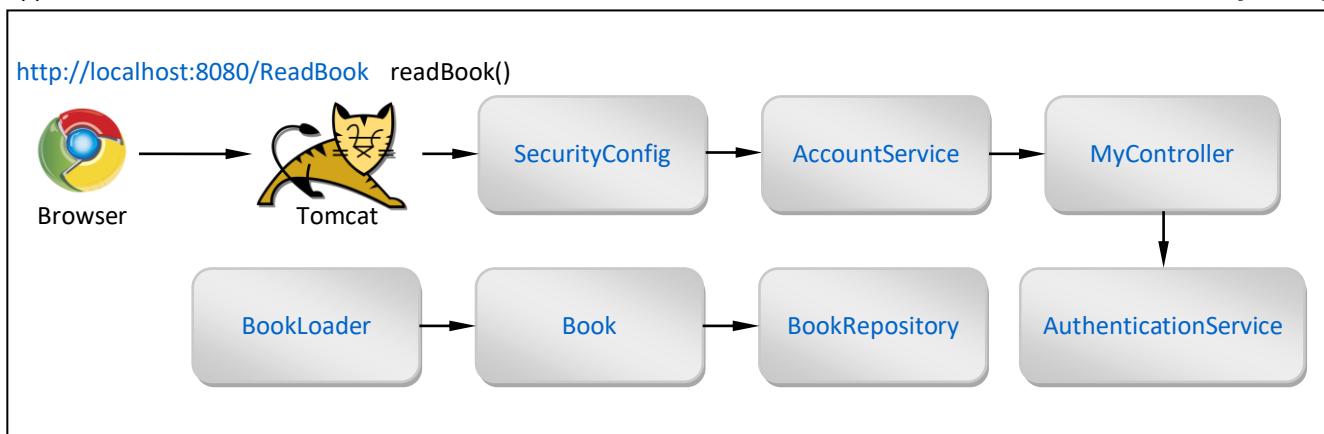
- This tutorial shows how to use **Custom Method** to Authorize access to Endpoint.
- We will create Custom Class `AuthenticationService.java` with Method `authenticate()` that returns
 - true** if Authenticated User owns the Book it wants to read
 - false** if Authenticated User doesn't own the Book it wants to read
- User will call <http://localhost:8080/ReadBook/1> where 1 represents bookId.
- We will load DB with few Books (as shown below).

MyController.java

```
@PreAuthorize("hasAuthority('book.read') AND @authenticationService.authenticate(authentication)")
```

Application Schema

[Results]



DB Table: Book

(preloaded data)

ID	TITLE	USERNAME
1	Book about dogs	john
2	Book about cats	bill

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	H2 Database	Enables in-memory H2 Database

Procedure

- **Create Project:** springboot_security_authorization_custom_books (add Spring Boot Starters from the table)
- **Edit File:** application.properties (specify Username, Password, Role)
- **Create Package:** **entities** (inside package main)
 - Create Class: PersonEntity.java (inside package entities)
- **Create Package:** **repositories** (inside package main)
 - Create Interface: PersonRepository.java (inside package repositories)
- **Create Package:** **config** (inside package main)
 - Create Class: BookLoader.java (inside package config)
 - Create Class: SecurityConfig.java (inside package config)
- **Create Package:** **services** (inside package main)
 - Create Class: AccountService.java (inside package services)
 - Create Class: AuthenticationService.java (inside package services)
- **Create Package:** **controllers** (inside package main)
 - Create Class: MyController.java (inside package controllers)

application.properties

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

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

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

Book.java

```
package com.ivoronline.springboot_security_authorization_custom_books.entities;

import org.springframework.stereotype.Component;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;

@Entity
@Component
public class Book {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    public Integer id;
    public String userName;
    public String title;

}
```

BookRepository.java

```
package com.ivoronline.springboot_security_authorization_custom_books.repositories;

import com.ivoronline.springboot_security_authorization_custom_books.entities.Book;
import org.springframework.data.repository.CrudRepository;

public interface BookRepository extends CrudRepository<Book, Integer> { }
```

BookLoader.java

```
package com.ivoronline.springboot_security_authorization_custom_books.config;

import com.ivoronline.springboot_security_authorization_custom_books.entities.Book;
import com.ivoronline.springboot_security_authorization_custom_books.repositories.BookRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.CommandLineRunner;
import org.springframework.stereotype.Component;
import org.springframework.transaction.annotation.Transactional;

@Component
public class BookLoader implements CommandLineRunner {

    @Autowired
    private BookRepository bookRepository;

    @Override
    @Transactional
    public void run(String... args) throws Exception {

        //BBOK1
        Book book1 = new Book();
        book1.title = "Book about dogs";
        book1.userName = "john";

        //BBOK2
        Book book2 = new Book();
        book2.title = "Book about cats";
        book2.userName = "bill";

        //STORE ACCOUNT INTO DB
        bookRepository.save(book1);
        bookRepository.save(book2);

    }
}
```

MyController.java

```
package com.ivoronline.springboot_security_authorization_custom_books.controllers;

import org.springframework.security.access.prepost.PreAuthorize;
import org.springframework.web.bind.annotation.PathVariable;
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("/ReadBook/{BookId}")
    @PreAuthorize("hasAuthority('book.read') AND @authenticationService.authenticate(authentication, #bookId)")
    public String readBook(@PathVariable("BookId") String bookId) {
        return "USER can read his Book";
    }

}
```

AuthenticationService.java

```
package com.ivoronline.springboot_security_authorization_custom_books.services;

import com.ivoronline.springboot_security_authorization_custom_books.entities.Book;
import com.ivoronline.springboot_security_authorization_custom_books.repositories.BookRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.core.Authentication;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.stereotype.Component;
import org.springframework.transaction.annotation.Transactional;

@Component
public class AuthenticationService {

    @Autowired
    BookRepository bookRepository;

    @Transactional
    public boolean authenticate(Authentication authentication, Integer bookId) {

        //GET BOOK
        Book book = bookRepository.findById(bookId).get();

        //GET BOOK USERNAME
        String bookUserName = book.userName;

        //GET AUTHENTICATED USERNAME
        UserDetails user = (UserDetails) authentication.getPrincipal();
        String authenticatedUserName = user.getUsername();

        //CHECK OWNERSHIP
        if (bookUserName.equals(authenticatedUserName)) { return true; }
        else { return false; }

    }

}
```

SecurityConfig.java

```
package com.ivoronline.springboot_security_authorization_custom_books.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.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
@EnableGlobalMethodSecurity(prePostEnabled = true)
public class SecurityConfig extends WebSecurityConfigurerAdapter {

    @Bean
    PasswordEncoder passwordEncoder() {
        return NoOpPasswordEncoder.getInstance();
    }

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

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

        //EVERYTHING ELSE IS LOCKED
        httpSecurity.formLogin();

    }

}
```

AccountService.java

```
package com.ivorononline.springboot_security_authorization_custom_books.services;

import org.springframework.beans.factory.annotation.Value;
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.Arrays;
import java.util.List;

@Service
public class AccountService implements UserDetailsService {

    //LOAD PROPERTIES (from application.properties file)
    @Value("${spring.security.user.name}") private String userName;
    @Value("${spring.security.user.password}") private String userPassword;
    @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 enteredUserName) throws UsernameNotFoundException {

        //CHECK USERNAME
        if(!enteredUserName.equals(userName)) { throw new UsernameNotFoundException("User not found"); }

        //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()));
        }

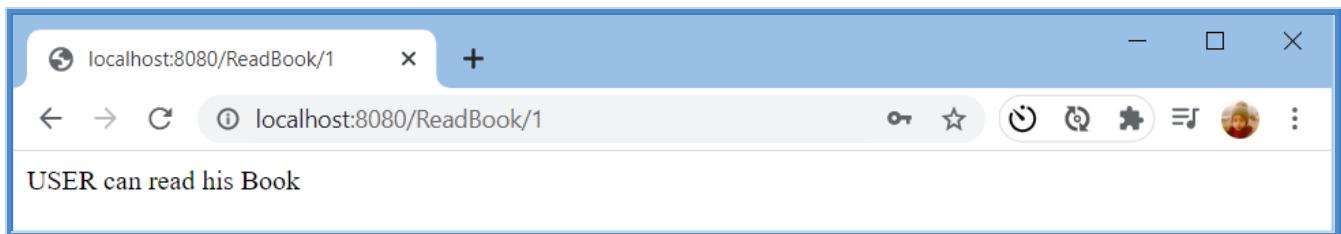
        //CREATE USER
        User user = new User(userName, userPassword, true, true, true, true, authorities);

        //RETURN USER
        return user;
    }
}
```

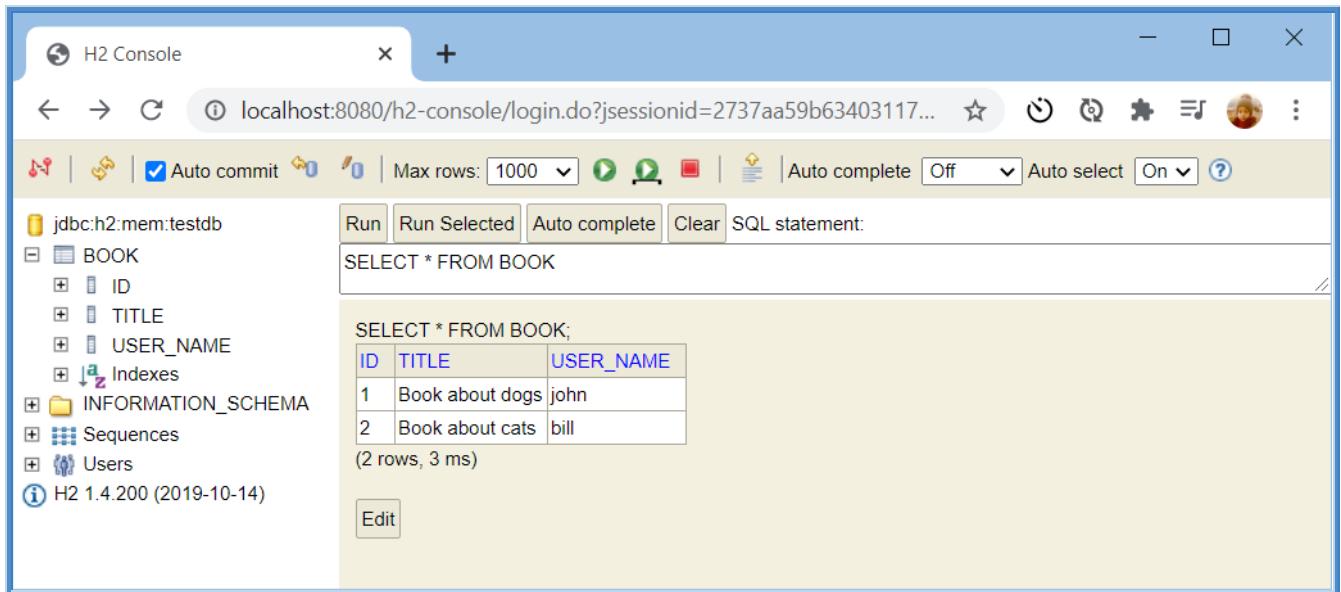
Results

<http://localhost:8080/ReadBook/1>

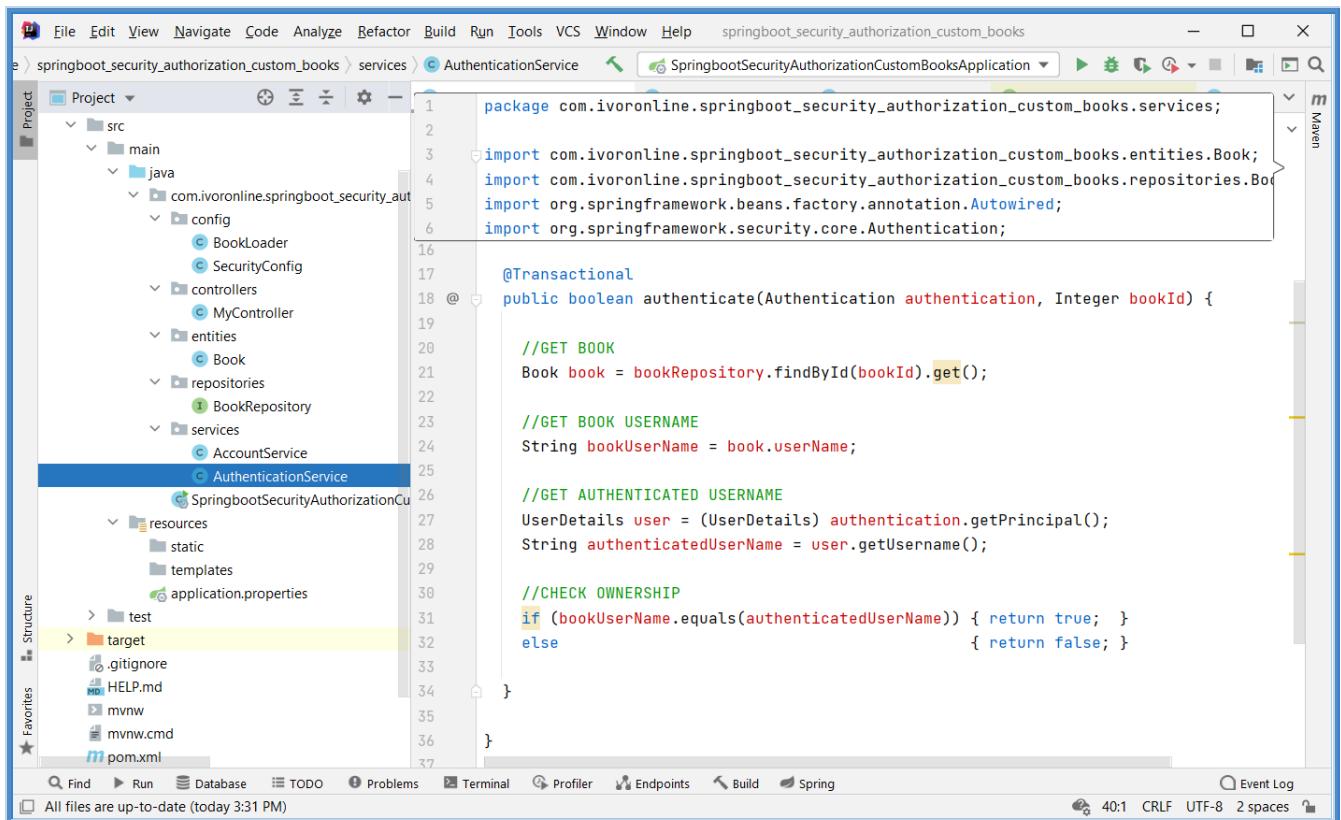
@PreAuthorize("hasAuthority('book.read') AND @authenticationService.authenticate(authentication)")



<http://localhost:8080/h2-console>



Application Structure



```
<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>

</dependencies>
```

1.4.6 URL Patterns - Ant Matchers

Info

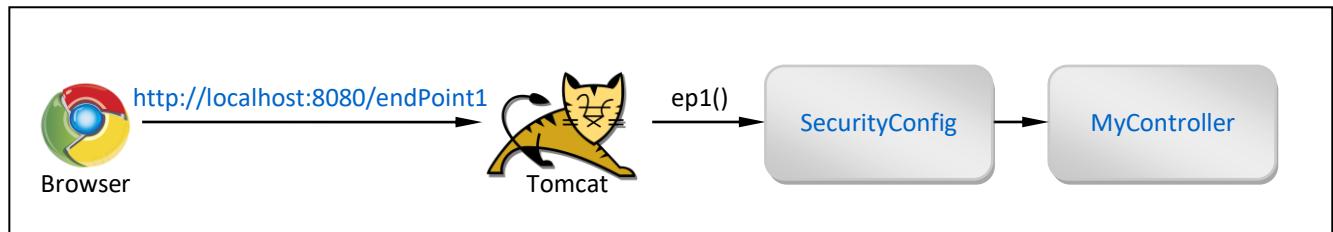
- When you add Spring Boot Starter **Security** every URL becomes unavailable unless you Log In.
- This tutorial shows how to loosen up this restriction by allowing access to certain URLs without having to Log In.
- We will do this by creating a Class **SecurityConfig** which extends and Overrides Method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

SecurityConfig.java

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

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

application.properties

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

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"; }
}
```

SecurityConfig.java

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

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;

@Configuration
@EnableWebSecurity
public class SecurityConfig extends WebSecurityConfigurerAdapter {

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

        //SPECIFY ACCESS TO ENDPOINTS
        httpSecurity.authorizeRequests()
            .antMatchers("/ endPoint1").denyAll()                                //No access (even after log in)
            .antMatchers("/ endPoint2").permitAll()                               //No log in (anonymous access)
            .antMatchers("/ endPoint3").hasRole("ADMIN")                         //ADMIN ROLE can access AFTER log in
            .antMatchers("/ endPoint4").hasAnyRole("ADMIN", "USER");           //ADMIN/USER ROLE can access AFTER log in

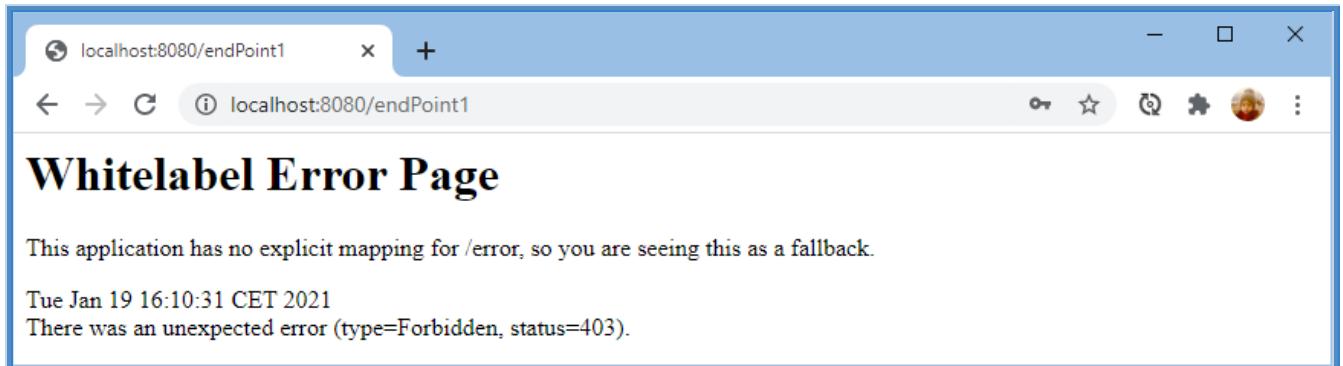
        //REDIRECT TO LOGIN FORM
        httpSecurity.formLogin();

    }
}
```

Results

<http://localhost:8080/endPoint1>

denyAll() => deny ALL after Login



localhost:8080/endPoint1

localhost:8080/endPoint1

Whitelabel Error Page

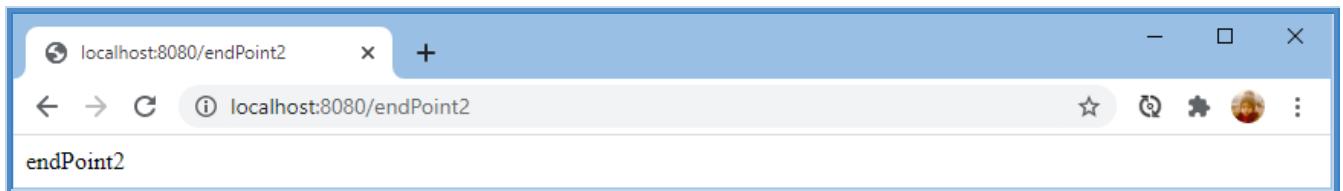
This application has no explicit mapping for /error, so you are seeing this as a fallback.

Tue Jan 19 16:10:31 CET 2021

There was an unexpected error (type=Forbidden, status=403).

<http://localhost:8080/endPoint2>

permitAll() => allow ALL without Login



localhost:8080/endPoint2

localhost:8080/endPoint2

endPoint2

<http://localhost:8080/endPoint3>

hasRole("ADMIN") => deny USER after Login



localhost:8080/endPoint3

localhost:8080/endPoint3

Whitelabel Error Page

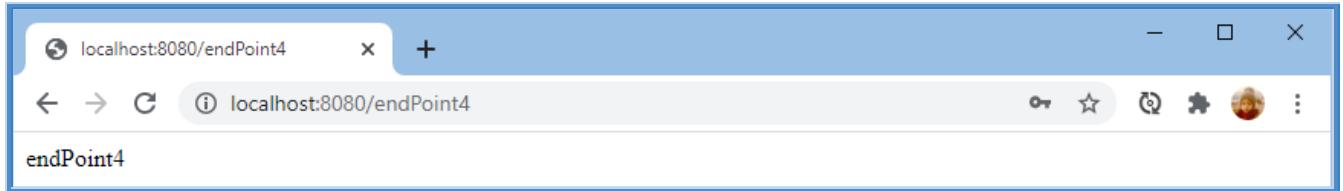
This application has no explicit mapping for /error, so you are seeing this as a fallback.

Tue Jan 19 16:12:31 CET 2021

There was an unexpected error (type=Forbidden, status=403).

<http://localhost:8080/endPoint4>

hasAnyRole("ADMIN", "USER") => allow USER after Login



localhost:8080/endPoint4

localhost:8080/endPoint4

endPoint4

Ant Matcher Examples

- Here are some additional examples that show how to exclude/include certain URLs from Authentication.
- Order in which `authorizeRequests()` are called is very important.

URL Patterns

PATTERN	EXAMPLES
<code>"/endPoint1"</code>	Specific URL
<code>"/endPoint1", "/endPoint2"</code>	List of URLs (patterns)
<code>"/*"</code>	All URLs on this level: <code>/hello, /endPoint123, /sublevel/endPoint4</code>
<code>"/**"</code>	All URLs on this or lower levels: <code>/hello, /endPoint123, /sublevel/endPoint4</code>
<code>"/end*"</code>	All URLs on this level that start with end: <code>/hello, /endPoint123, /sublevel/endPoint4</code>
<code>HttpMethod.GET, "/end*"</code>	All End Points that accept GET and start with /end (@RequestMapping, @GetMapping)

SecurityConfig.java

(deny/permit)

```
@Override  
protected void configure(HttpSecurity httpSecurity) throws Exception {  
    httpSecurity  
        .authorizeRequests(authorize -> {  
            authorize.antMatchers("/endPoint1").denyAll(); //Deny  
            authorize.antMatchers("/endPoint2").permitAll(); //Permit  
            authorize.antMatchers("/endPoint3").hasRole("ADMIN"); //Role  
        })  
        .authorizeRequests().anyRequest().authenticated()  
        .and().formLogin()  
        .and().httpBasic();  
}
```

SecurityConfig.java

(specify list of URLs)

```
@Override  
protected void configure(HttpSecurity httpSecurity) throws Exception {  
    httpSecurity  
        .authorizeRequests(authorize -> { authorize.antMatchers("/endPoint1", "/endPoint2").permitAll(); })  
        .authorizeRequests().anyRequest().authenticated()  
        .and().formLogin()  
        .and().httpBasic();  
}
```

SecurityConfig.java

(* wildcard: any URL on this level)

```
@Override  
protected void configure(HttpSecurity httpSecurity) throws Exception {  
    httpSecurity  
        .authorizeRequests(authorize -> { authorize.antMatchers("/*").permitAll(); }) //EP: 1, 2, 3  
        .authorizeRequests().anyRequest().authenticated()  
        .and().formLogin()  
        .and().httpBasic();  
}
```

```

@Override
protected void configure(HttpSecurity httpSecurity) throws Exception {
    httpSecurity
        .authorizeRequests(authorize -> { authorize.antMatchers("/**").permitAll(); }) //EP: 1, 2, 3, 4
        .authorizeRequests().anyRequest().authenticated()
        .and().formLogin()
        .and().httpBasic();
}

```

```

@Override
protected void configure(HttpSecurity httpSecurity) throws Exception {
    httpSecurity
        .authorizeRequests(authorize -> { authorize.antMatchers("/end*").permitAll(); })
        .authorizeRequests().anyRequest().authenticated()
        .and().formLogin()
        .and().httpBasic();
}

```

```

@Override
protected void configure(HttpSecurity httpSecurity) throws Exception {
    httpSecurity
        .authorizeRequests(authorize -> { authorize.antMatchers(HttpMethod.GET, "/end*").permitAll(); })
        .authorizeRequests().anyRequest().authenticated()
        .and().formLogin()
        .and().httpBasic();
}

```

1.4.7 Roles

Info

[G]

- This tutorial shows how to use Roles to Authorize access to Endpoints.
- This is simple out of the box approach that requires no custom code.
- Authorization is configured inside the Controller by using simple Annotations `@Secured({"ROLE_ADMIN", "ROLE_USER"})`.

SecurityConfig.java

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

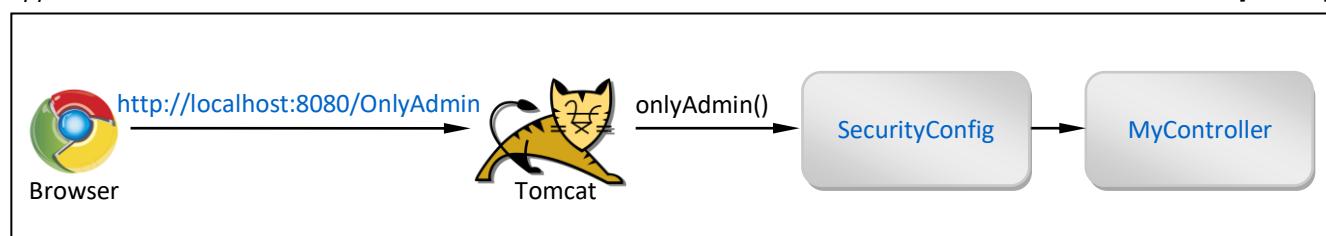
MyController.java

(`@Secured` can only be used with Roles)

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

Application Schema

[Results]



Spring Boot Starters

GROUP	DEPENDENCY	DESCRIPTION
Web	Spring Web	Enables <code>@Controller</code> , <code>@RequestMapping</code> and Tomcat Server
Security	Spring Security	Enables Spring Security

Procedure

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

application.properties

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

SecurityConfig.java

```
package com.ivoronline.springboot_security_authorization_roles.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.EnableWebSecurity;
import org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;

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

    @Override
    protected void configure(HttpSecurity httpSecurity) throws Exception {
        httpSecurity.formLogin();
    }

}
```

MyController.java

```
package com.ivoronline.springboot_security_authorization_roles.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 {

    @ResponseBody
    @Secured("ROLE_ADMIN")
    @RequestMapping("/OnlyAdmin")
    public String onlyAdmin() {
        return "Only ROLE_ADMIN";
    }

    @ResponseBody
    @Secured({"ROLE_ADMIN", "ROLE_USER"})
    @RequestMapping("/AdminAndUser")
    public String adminAndUser() {
        return "ROLE_ADMIN and ROLE_USER";
    }

}
```

Results

<http://localhost:8080/OnlyAdmin>

`@Secured("ROLE_ADMIN") => deny USER after Login`

The screenshot shows a browser window with the URL `localhost:8080/OnlyAdmin`. The page title is "Whitelabel Error Page". The content area displays the message: "This application has no explicit mapping for /error, so you are seeing this as a fallback." Below this, it shows the timestamp "Wed Jan 20 15:29:38 CET 2021" and the error message "There was an unexpected error (type=Forbidden, status=403)." The browser interface includes standard navigation buttons (back, forward, search) and a toolbar with icons for refresh, run, and other developer tools.

<http://localhost:8080/endPoint2>

`@Secured({"ROLE_ADMIN", "ROLE_USER"}) => allow USER after Login`

The screenshot shows a browser window with the URL `localhost:8080/AdminAndUser`. The content area displays the message "ROLE_ADMIN and ROLE_USER". The browser interface is identical to the one in the previous screenshot, showing standard navigation and developer tool icons.

1.4.8 Authorities - application.properties

Info

[G]

- This tutorial shows how to use Authorities to Authorize access to Endpoints.
User and Profiles (their authorities) are defined in [application.properties](#) (for simplicity so that we don't need to use DB).
- The main part of this tutorial is [@Service class AccountService implements UserDetailsService { .. }.](#)
It [@Overrides loadUserByUsername\(String username\)](#) which is automatically called by Login Form with entered username
Inside this method we create [User Object](#) with authorities specified by [spring.security.user.profile = USER](#).

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
profile.admin = book.create, book.read, book.update, book.delete
```

SecurityConfig.java

To enable @PreAuthorize

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

MyController.java

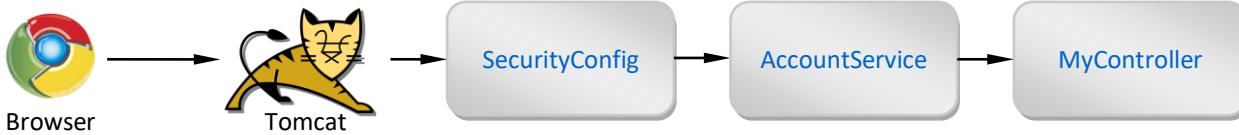
CRUD DB Operations

```
@PreAuthorize("hasAuthority('book.create')")
@PreAuthorize("hasAuthority('book.read')")
@PreAuthorize("hasAuthority('book.update')")
@PreAuthorize("hasAuthority('book.delete')")
```

Application Schema

[Results]

<http://localhost:8080/ReadBook> readBook()



Spring Boot Starters

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

Procedure

- [Create Project:](#) springboot_security_authorizationAuthorities (add Spring Boot Starters from the table)
Edit File: application.properties (specify Username, Password, Role)
- [Create Package:](#) controllers (inside main package)
- [Create Class:](#) MyController.java (inside controllers package)
- [Create Package:](#) config (inside main package)
- [Create Class:](#) SecurityConfig.java (inside config package)
- [Create Package:](#) services (inside main package)
- [Create Class:](#) AccountService.java (inside services package)

application.properties

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

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

AccountService.java

```
package com.example.springboot_security_authorization_authorities.services;

import org.springframework.beans.factory.annotation.Value;
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.Arrays;
import java.util.List;

@Service
public class AccountService 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", true, true, true, authorities);

        //RETURN USER
        return user;
    }
}
```

SecurityConfig.java

```
package com.example.springboot_security_authorizationAuthorities.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.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
@EnableGlobalMethodSecurity(prePostEnabled = true)
public class SecurityConfig extends WebSecurityConfigurerAdapter {

    @Bean
    PasswordEncoder passwordEncoder() {
        return NoOpPasswordEncoder.getInstance();
    }

    @Override
    protected void configure(HttpSecurity httpSecurity) throws Exception {
        httpSecurity.formLogin();
    }
}
```

MyController.java

(Authorities represent DB **CRUD** operations)

```
package com.example.springboot_security_authorizationAuthorities.controllers;

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

@Controller
public class MyController {

    @ResponseBody
    @PreAuthorize("hasAuthority('book.create')")
    @RequestMapping("/CreateBook")
    public String createBook() {
        return "Only ADMIN can create Book"; }

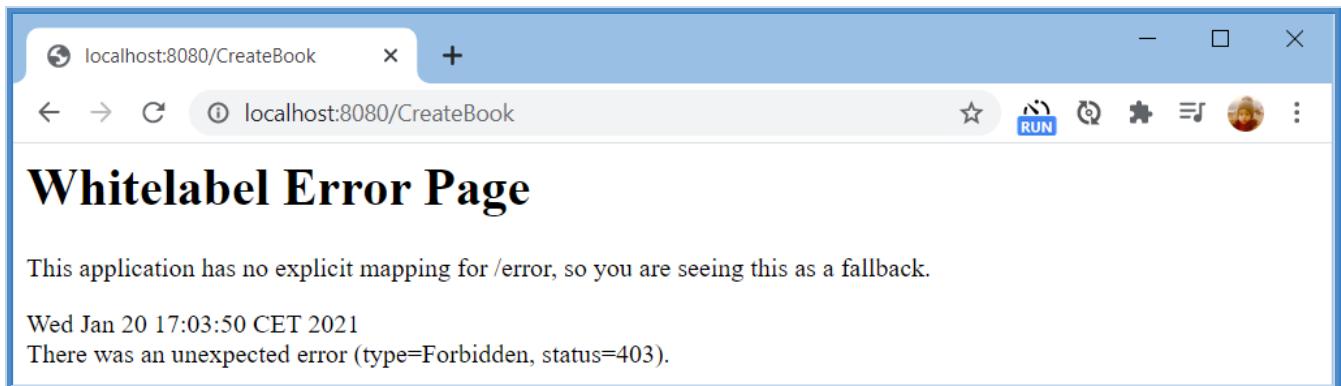
    @ResponseBody
    @PreAuthorize("hasAuthority('book.read')")
    @RequestMapping("/ReadBook")
    public String readBook() {
        return "ADMIN and USER can read Book";
    }

    @ResponseBody
    @PreAuthorize("hasAuthority('book.update')")
    @RequestMapping("/UpdateBook")
    public String updateBook() {
        return "Only ADMIN can update Book";
    }
}
```

Results

<http://localhost:8080/CreateBook>

`@PreAuthorize("hasAuthority('book.create')") => deny USER after Login`



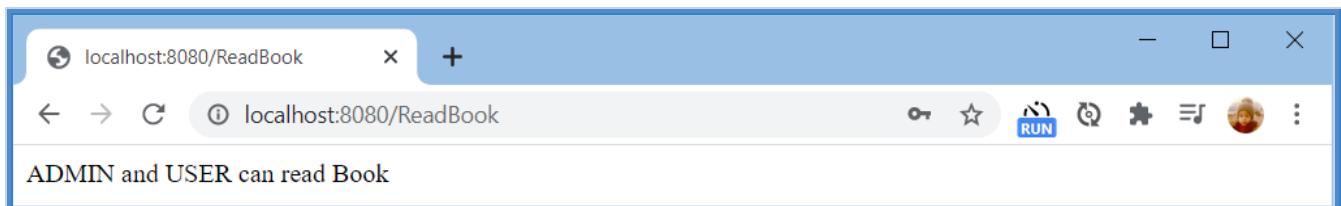
This application has no explicit mapping for /error, so you are seeing this as a fallback.

Wed Jan 20 17:03:50 CET 2021

There was an unexpected error (type=Forbidden, status=403).

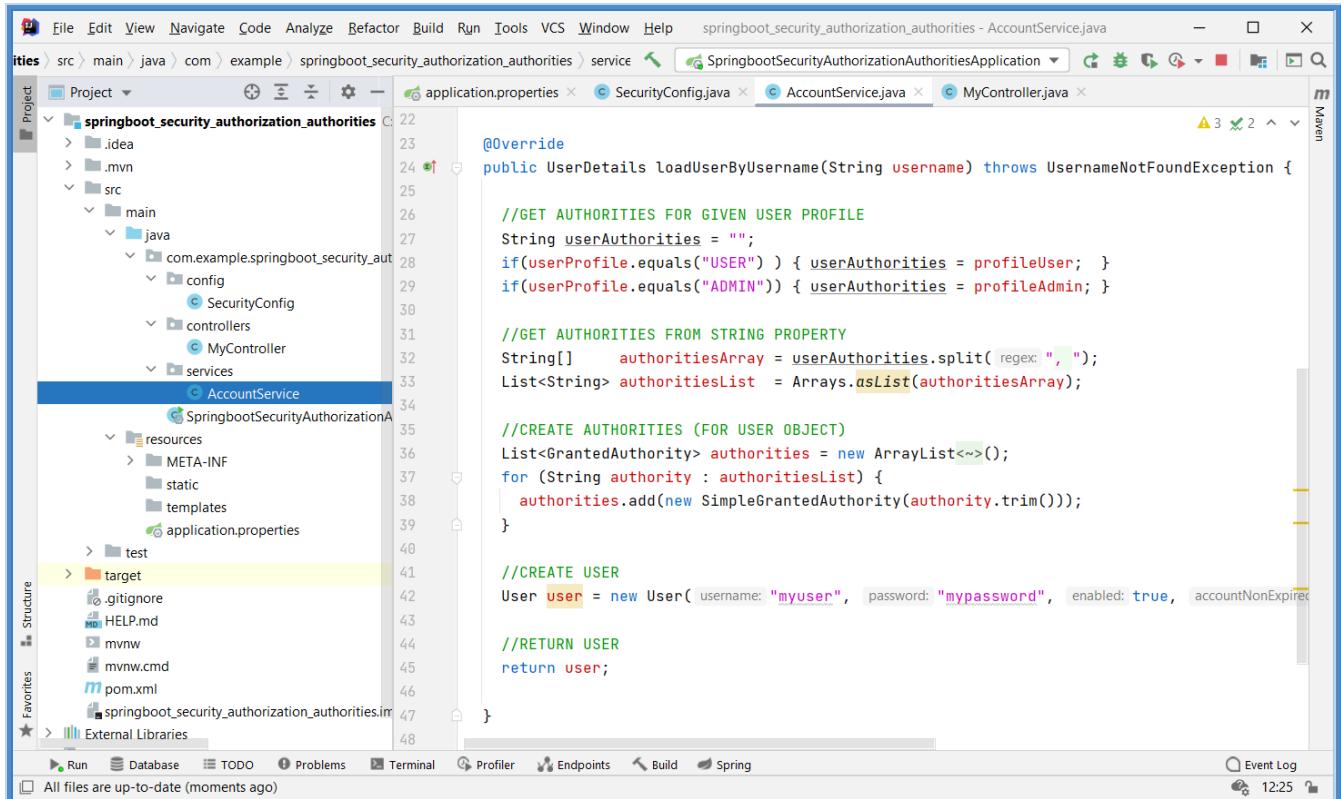
<http://localhost:8080/ReadBook>

`@PreAuthorize("hasAuthority('book.read')") => allow USER after Login`



ADMIN and USER can read Book

Application Structure



```
22
23
24 @Override
25
26     public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {
27
28         //GET AUTHORITIES FOR GIVEN USER PROFILE
29         String userAuthorities = "";
30         if(userProfile.equals("USER")) { userAuthorities = profileUser; }
31         if(userProfile.equals("ADMIN")) { userAuthorities = profileAdmin; }
32
33         //GET AUTHORITIES FROM STRING PROPERTY
34         String[] authoritiesArray = userAuthorities.split( regex: ", " );
35         List<String> authoritiesList = Arrays.asList(authoritiesArray);
36
37         //CREATE AUTHORITIES (FOR USER OBJECT)
38         List<GrantedAuthority> authorities = new ArrayList<>();
39         for (String authority : authoritiesList) {
40             authorities.add(new SimpleGrantedAuthority(authority.trim()));
41         }
42
43         //CREATE USER
44         User user = new User( username: "myuser", password: "mypassword", enabled: true, accountNonExpired: true, accountNonLocked: true, credentialsNonExpired: true );
45
46         //RETURN USER
47         return user;
48     }

```

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>
```

1.4.9 Authorities - DB

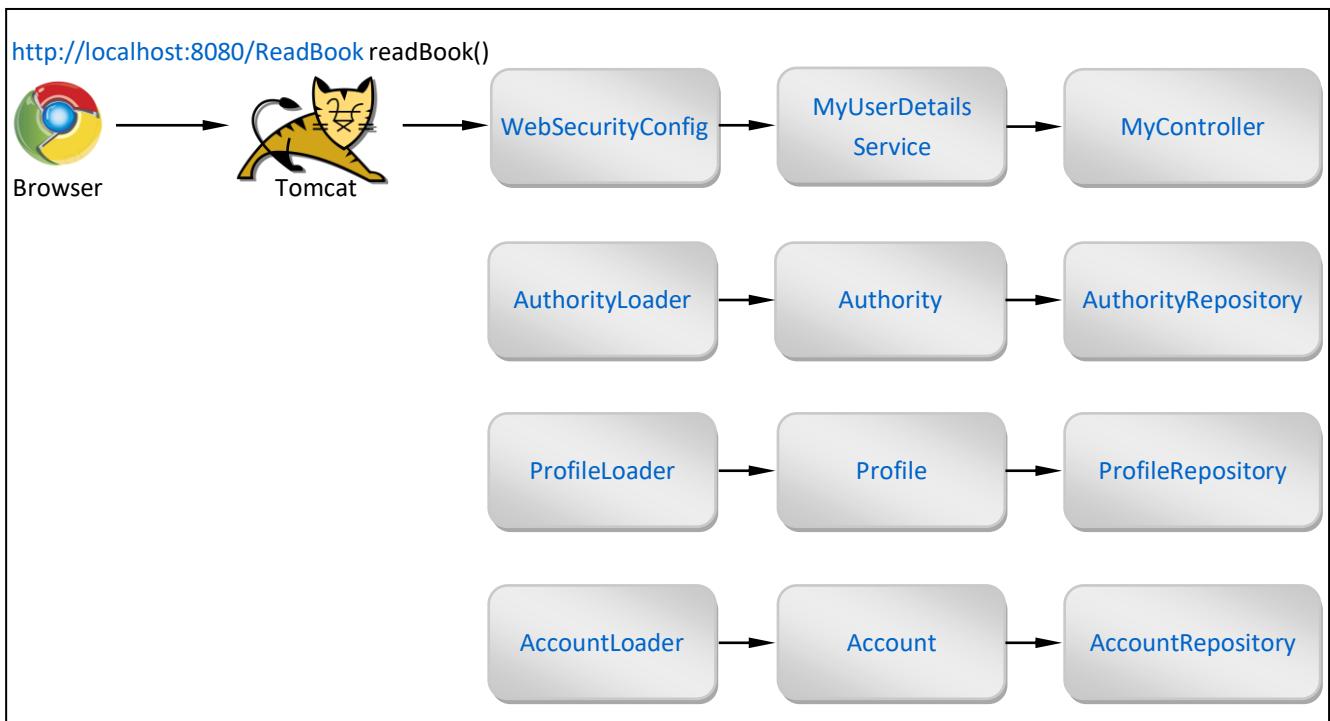
Info

[G]

- This tutorial shows how to get User's Authorities from the DB by grouping them into Profiles.
- By assigning Profile to Account, Account is given Authorities that are related to that Profile.

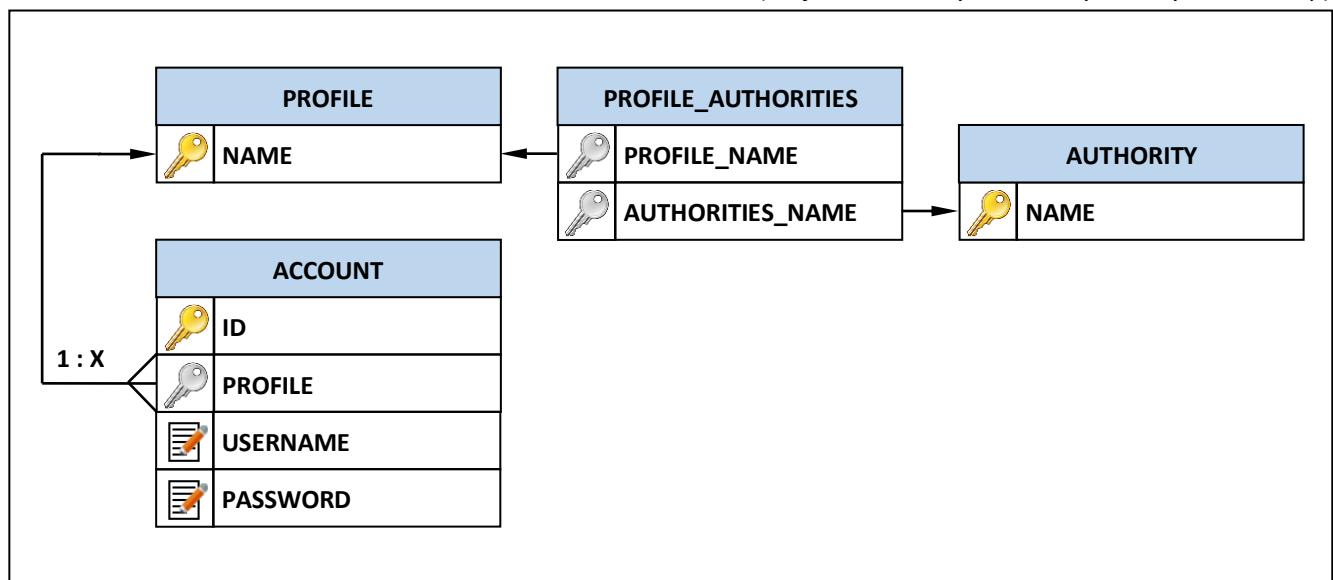
Application Schema

[Results]



DB Schema

(Profile & Authority have ManyToMany Relationship)



ACCOUNT

(Loaded Data)

ID	USERNAME	PASSWORD	PROFILE
1	admin	adminpassword	ADMIN
2	user	userpassword	USER

PROFILE

NAME
ADMIN
USER

PROFILE_AUTHORITIES

PROFILE_NAME	AUTHORITIES_NAME
ADMIN	book.create
ADMIN	book.delete
ADMIN	book.read
ADMIN	book.update
USER	book.read

AUTHORITY

NAME
book.create
book.delete
book.read
book.update

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, @Id
SQL	H2 Database	Enables: in-memory H2 Database

Overview

- The main part of this tutorial is `@Service` class `AccountService` implements `UserDetailsService` { ... }.
- It `@Overrides loadUserByUsername(String username)` which is automatically called by Login Form with entered username. Inside this method we create `User` Object with authorities which are related to Profile that is given to Account.

SecurityConfig.java

To enable `@PreAuthorize`

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

MyController.java

CRUD DB Operations

```
@PreAuthorize("hasAuthority('book.create')")
@PreAuthorize("hasAuthority('book.read')")
@PreAuthorize("hasAuthority('book.update')")
@PreAuthorize("hasAuthority('book.delete')")
```

Procedure

- Create Project: `springboot_security_authorizationAuthorities_db` (add Spring Boot Starters from the table)
- Edit File: `application.properties` (specify Username, Password, Role)
- Create Package: `entities` (inside main package)
 - Create Class: `Authority.java` (inside entities package)
 - Create Class: `Profile.java` (inside entities package)
 - Create Class: `Account.java` (inside entities package)
- Create Package: `repositories` (inside main package)
 - Create Class: `AuthorityRepository.java` (inside repositories package)
 - Create Class: `ProfileRepository.java` (inside repositories package)
 - Create Class: `AccountRepository.java` (inside repositories package)
- Create Package: `config` (inside main package)
 - Create Class: `WebSecurityConfig.java` (inside config package)
- Create Package: `startup` (inside main package)
 - Create Class: `AuthorityLoader.java` (inside config package)
 - Create Class: `ProfileLoader.java` (inside config package)
 - Create Class: `AccountLoader.java` (inside config package)
- Create Package: `services` (inside main package)
 - Create Class: `MyUserDetailsService.java` (inside services package)
- Create Package: `controllers` (inside main package)
 - Create Class: `MyController.java` (inside controllers package)

application.properties

```
# H2 CONSOLE
spring.h2.console.enabled = true
spring.datasource.url      = jdbc:h2:mem:testdb
```

Authority.java

```
package com.ivoronline.springboot_security_authorizationAuthorities_db.entities;

import javax.persistence.Entity;
import javax.persistence.Id;

@Entity
public class Authority {

    @Id
    public String name;

    //CONSTRUCTORS
    public Authority() { }                                //Forced by @Entity
    public Authority(String name) { this.name = name; }   //To simplify AuthorityLoader

}
```

Profile.java

```
package com.ivoronline.springboot_security_authorizationAuthorities_db.entities;

import javax.persistence.CascadeType;
import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.JoinColumn;
import javax.persistence.ManyToMany;
import javax.persistence.OneToMany;
import java.util.HashSet;
import java.util.Set;

@Entity
public class Profile {

    @Id
    public String name;

    @OneToMany(cascade = CascadeType.ALL)
    @JoinColumn(name = "profile")
    public Set<Account> account = new HashSet<>();

    @ManyToMany(cascade = CascadeType.ALL)
    public Set<Authority> authorities = new HashSet<>();

}
```

Account.java

```
package com.ivoronline.springboot_security_authorizationAuthorities_db.entities;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;

@Entity
public class Account {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    public Integer id;
    public String username;
    public String password;

    //FOREIGN KEY
    public String profile;

}
```

AuthorityRepository.java

```
package com.ivorononline.springboot_security_authorizationAuthorities_db.repositories;

import com.ivorononline.springboot_security_authorizationAuthorities_db.entities.Authority;
import org.springframework.data.jpa.repository.JpaRepository;

public interface AuthorityRepository extends JpaRepository<Authority, String> { }
```

ProfileRepository.java

```
package com.ivorononline.springboot_security_authorizationAuthorities_db.repositories;

import com.ivorononline.springboot_security_authorizationAuthorities_db.entities.Profile;
import org.springframework.data.jpa.repository.JpaRepository;

public interface ProfileRepository extends JpaRepository<Profile, String> { }
```

AccountRepository.java

```
package com.ivorononline.springboot_security_authorizationAuthorities_db.repositories;

import com.ivorononline.springboot_security_authorizationAuthorities_db.entities.Account;
import org.springframework.data.jpa.repository.JpaRepository;

public interface AccountRepository extends JpaRepository<Account, Integer> {
    Account findByUsername(String Username);
}
```

AuthorityLoader.java

```
package com.ivorononline.springboot_security_authorizationAuthorities_db.startup;

import com.ivorononline.springboot_security_authorizationAuthorities_db.entities.Authority;
import com.ivorononline.springboot_security_authorizationAuthorities_db.repositories.AuthorityRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.CommandLineRunner;
import org.springframework.core.annotation.Order;
import org.springframework.stereotype.Component;
import org.springframework.transaction.annotation.Transactional;

@Component
public class AuthorityLoader implements CommandLineRunner {

    @Autowired private AuthorityRepository authorityRepository;

    @Override
    @Transactional
    public void run(String... args) throws Exception {
        authorityRepository.save(new Authority("book.create"));
        authorityRepository.save(new Authority("book.read"));
        authorityRepository.save(new Authority("book.update"));
        authorityRepository.save(new Authority("book.delete"));
    }
}
```

ProfileLoader.java

```
package com.ivorononline.springboot_security_authorizationAuthorities_db.startup;

import com.ivorononline.springboot_security_authorizationAuthorities_db.entities.Authority;
import com.ivorononline.springboot_security_authorizationAuthorities_db.entities.Profile;
import com.ivorononline.springboot_security_authorizationAuthorities_db.repositories.AuthorityRepository;
import com.ivorononline.springboot_security_authorizationAuthorities_db.repositories.ProfileRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.CommandLineRunner;
import org.springframework.core.annotation.Order;
import org.springframework.stereotype.Component;
import org.springframework.transaction.annotation.Transactional;

@Component
@Order(2)
public class ProfileLoader implements CommandLineRunner {

    @Autowired private ProfileRepository profileRepository;
    @Autowired private AuthorityRepository authorityRepository;

    @Override
    @Transactional
    public void run(String... args) throws Exception {

        //GET AUTHORITIES.
        Authority bookCreate = authorityRepository.findById("book.create").get();
        Authority bookRead   = authorityRepository.findById("book.read").get();
        Authority bookUpdate = authorityRepository.findById("book.update").get();
        Authority bookDelete = authorityRepository.findById("book.delete").get();

        //USER
        Profile user      = new Profile();
        user.name = "USER";
        user.authorities.add(bookRead);

        //ADMIN
        Profile admin     = new Profile();
        admin.name = "ADMIN";
        admin.authorities.add(bookCreate);
        admin.authorities.add(bookRead);
        admin.authorities.add(bookUpdate);
        admin.authorities.add(bookDelete);

        //STORE PROFILES INTO DB
        profileRepository.save(user);
        profileRepository.save(admin);

    }
}
```

AccountLoader.java

```
package com.ivorononline.springboot_security_authorizationAuthorities_db.startup;

import com.ivorononline.springboot_security_authorizationAuthorities_db.entities.Account;
import com.ivorononline.springboot_security_authorizationAuthorities_db.repositories.AccountRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.CommandLineRunner;
import org.springframework.core.annotation.Order;
import org.springframework.stereotype.Component;
import org.springframework.transaction.annotation.Transactional;

@Component
@Order(3)
public class AccountLoader implements CommandLineRunner {

    @Autowired private AccountRepository accountRepository;

    @Override
    @Transactional
    public void run(String... args) throws Exception {

        //ADMIN: ANDREW
        Account admin = new Account();
        admin.username = "admin";
        admin.password = "adminpassword";
        admin.profile = "ADMIN";

        //USER: URBAN
        Account user = new Account();
        user.username = "user";
        user.password = "userpassword";
        user.profile = "USER";

        //STORE ACCOUNT INTO DB
        accountRepository.save(admin);
        accountRepository.save(user);

    }
}
```

WebSecurityConfig.java

```
package com.ivoronline.springboot_security_authorizationAuthorities_db.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.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
@EnableGlobalMethodSecurity(prePostEnabled = true)
public class WebSecurityConfig extends WebSecurityConfigurerAdapter {

    @Bean
    PasswordEncoder passwordEncoder() {
        return NoOpPasswordEncoder.getInstance();
    }

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

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

        //LOCK EVERYTHING ELSE (BEHIND LOG IN)
        httpSecurity.formLogin();

    }

}
```

MyUserDetailsService.java

```
package com.ivorononline.springboot_security_authorizationAuthorities_db.services;

import com.ivorononline.springboot_security_authorizationAuthorities_db.entities.Account;
import com.ivorononline.springboot_security_authorizationAuthorities_db.entities.Authority;
import com.ivorononline.springboot_security_authorizationAuthorities_db.entities.Profile;
import com.ivorononline.springboot_security_authorizationAuthorities_db.repositories.AccountRepository;
import com.ivorononline.springboot_security_authorizationAuthorities_db.repositories.ProfileRepository;
import org.springframework.beans.factory.annotation.Autowired;
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 org.springframework.transaction.annotation.Transactional;
import java.util.ArrayList;
import java.util.List;

@Service
public class MyUserDetailsService implements UserDetailsService {

    @Autowired
    AccountRepository accountRepository;

    @Autowired
    private ProfileRepository profileRepository;

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

        //GET ACCOUNT
        Account account = accountRepository.findByUsername(username);

        //GET PROFILE (WITH AUTHORITIES)
        Profile profile = profileRepository.findById(account.profile).get();

        //CREATE AUTHORITIES (TO CREATE USER)
        List<GrantedAuthority> authorities = new ArrayList<GrantedAuthority>();
        for(Authority authority : profile.authorities) {
            authorities.add(new SimpleGrantedAuthority(authority.name));
        }

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

        //RETURN USER
        return user;
    }
}
```

```
package com.ivoronline.springboot_security_authorization_authorities_db.controllers;

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

@Controller
public class MyController {

    @ResponseBody
    @PreAuthorize("hasAuthority('book.create')")
    @RequestMapping("/CreateBook")
    public String createBook() {
        return "Only ADMIN can create Book";
    }

    @ResponseBody
    @PreAuthorize("hasAuthority('book.read')")
    @RequestMapping("/ReadBook")
    public String readBook() {
        return "ADMIN and USER can read Book";
    }

    @ResponseBody
    @PreAuthorize("hasAuthority('book.update')")
    @RequestMapping("/UpdateBook")
    public String updateBook() {
        return "Only ADMIN can update Book";
    }

    @ResponseBody
    @PreAuthorize("hasAuthority('book.delete')")
    @RequestMapping("/DeleteBook")
    public String deleteBook() {
        return "Only ADMIN can delete Book";
    }
}
```

Results

<http://localhost:8080/CreateBook>

@PreAuthorize("hasAuthority('book.create')") => deny **USER** after Login



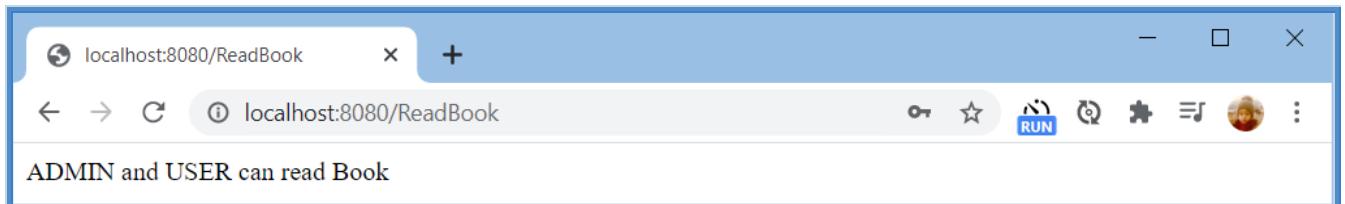
This application has no explicit mapping for /error, so you are seeing this as a fallback.

Wed Jan 20 17:03:50 CET 2021

There was an unexpected error (type=Forbidden, status=403).

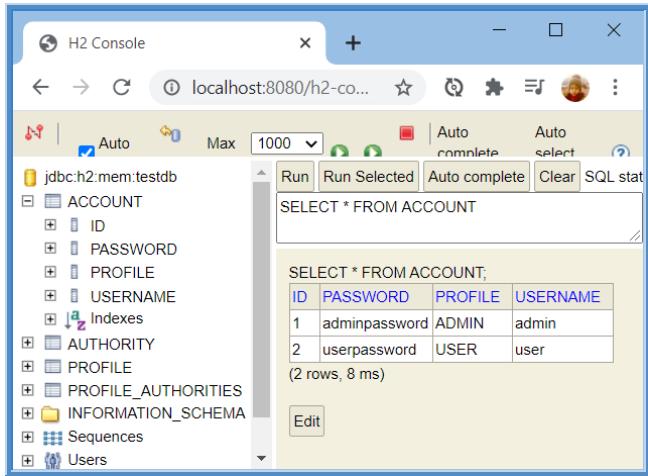
<http://localhost:8080/ReadBook>

@PreAuthorize("hasAuthority('book.read')") => allow **USER** after Login



ADMIN and USER can read Book

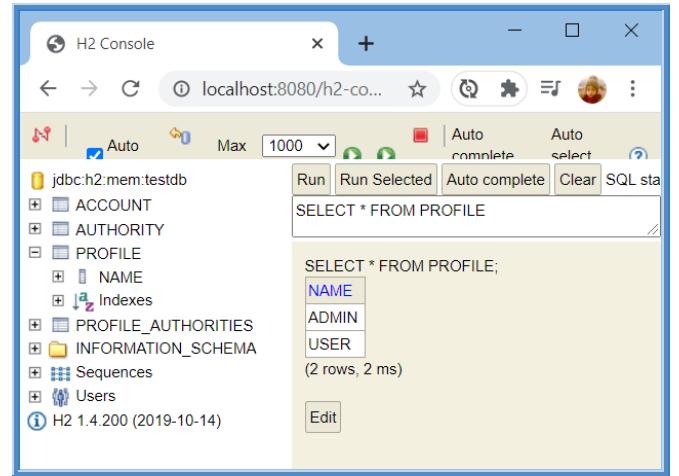
ACCOUNT



SELECT * FROM ACCOUNT;

ID	PASSWORD	PROFILE	USERNAME
1	adminpassword	ADMIN	admin
2	userpassword	USER	user

PROFILE

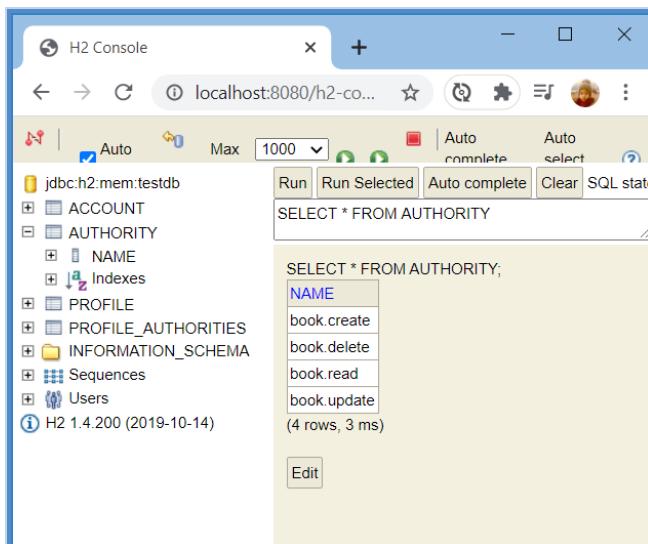


SELECT * FROM PROFILE;

NAME
ADMIN
USER

<http://localhost:8080/h2-console>

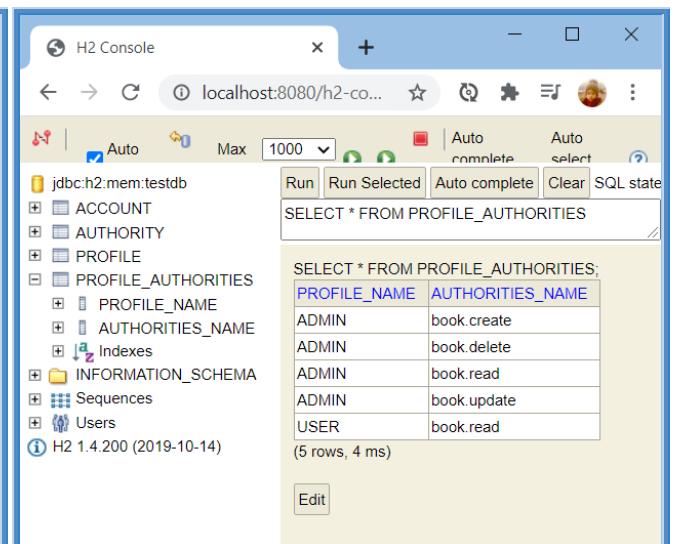
AUTHORITY



SELECT * FROM AUTHORITY;

NAME
book.create
book.delete
book.read
book.update

PROFILE_AUTHORITIES



SELECT * FROM PROFILE_AUTHORITIES;

PROFILE_NAME	AUTHORITIES_NAME
ADMIN	book.create
ADMIN	book.delete
ADMIN	book.read
ADMIN	book.update
USER	book.read

Application Structure

The screenshot shows the IntelliJ IDEA interface with the following details:

- Project Tree:** The project is named "springboot_security_authorizationAuthorities_db". It contains a "src" directory with "main" and "java" sub-directories. The "java" directory contains packages like "com.ivoronline.springboot_security", "controllers", "entities", "repositories", and "services". The "services" package contains a class "MyUserDetailsService".
- Code Editor:** The code for "MyUserDetailsService.java" is displayed. It implements the "UserDetailsService" interface and uses "AccountRepository" and "ProfileRepository" via autowiring. It overrides the "loadUserByUsername" method to return a "User" object. It also handles creating authorities and users.
- Toolbars and Status Bar:** The status bar at the bottom shows "Pushed 2 commits to origin/master (6 minutes ago)" and other build-related information.

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>

</dependencies>
```

1.5 Password Encoders

Info

- Password Encoders are also known as Hash Algorithms.
- Password Encoders are used to store encoded Password into DB (to protect them if somebody gets access to DB).
- When User wants to Login/Authenticate he will enter Raw Password.
- Spring then needs to compare Entered Raw Password with Stored Encoded Password.
- Comparison is done by using `matches()` Method to compare Passwords
 - taking Entered Raw Password
 - apply encoding algorithm
 - using `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"); }
```

1.5.1 No Operation

Info

[G]

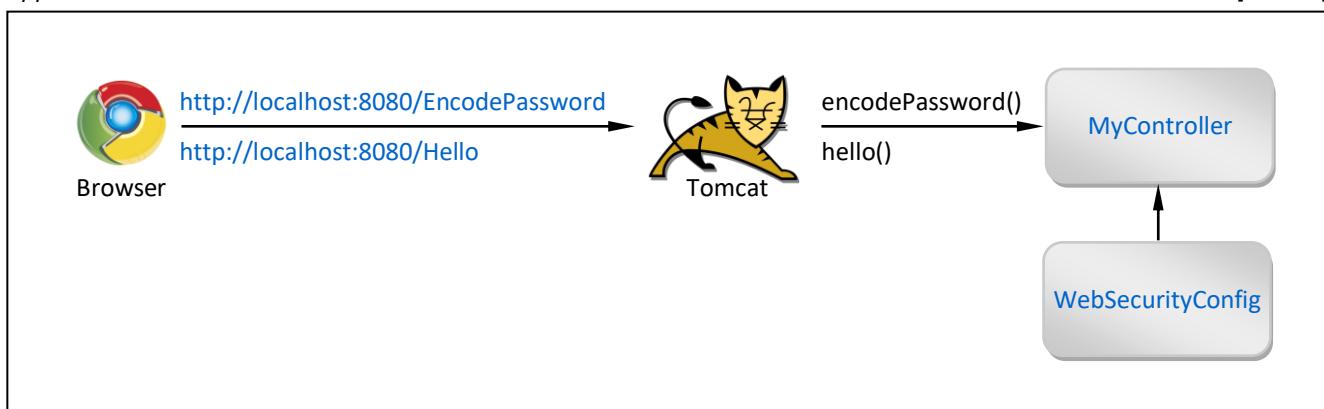
- No Operation Password Encoder doesn't actually encode Password at all.
Instead it leaves it as it is in its original String format.
But when you have Passwords that are not encoded you need to specify this encoder so that Spring would know how to properly read the Passwords (that is without decoding 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 <code>@Controller</code> , <code>@RequestMapping</code> 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
    //=====

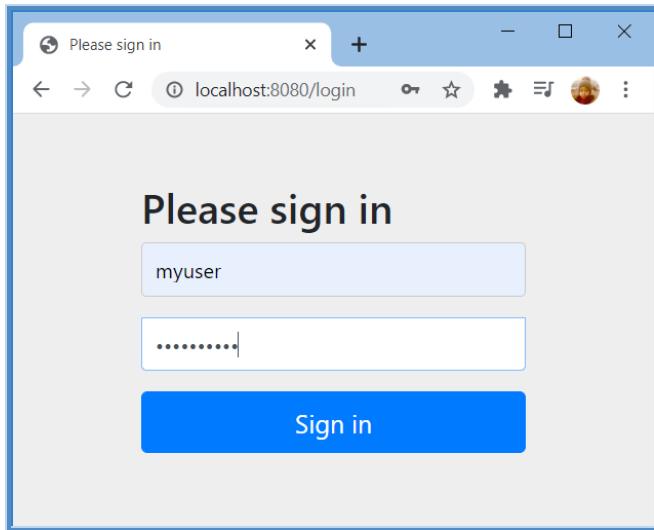
    @Override
    protected void configure(HttpSecurity httpSecurity) throws Exception {
        httpSecurity.authorizeRequests().antMatchers("/EncodePassword").permitAll(); //Anonymous Access
        httpSecurity.authorizeRequests().anyRequest().authenticated(); //Authenticated Access
        httpSecurity.formLogin();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

    }
}
```

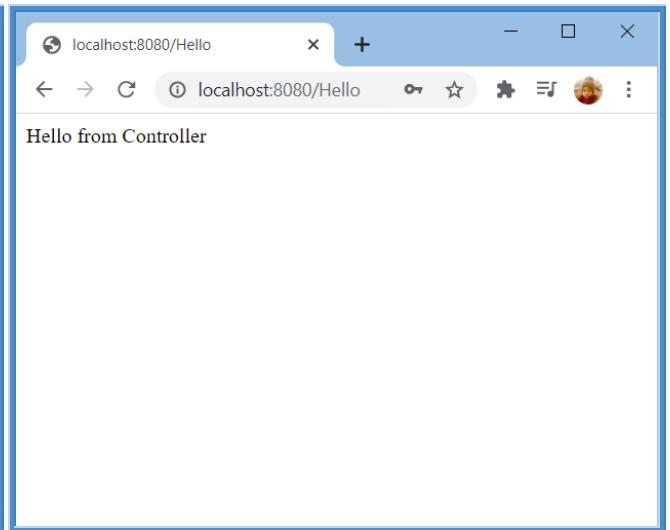
Results

- <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>

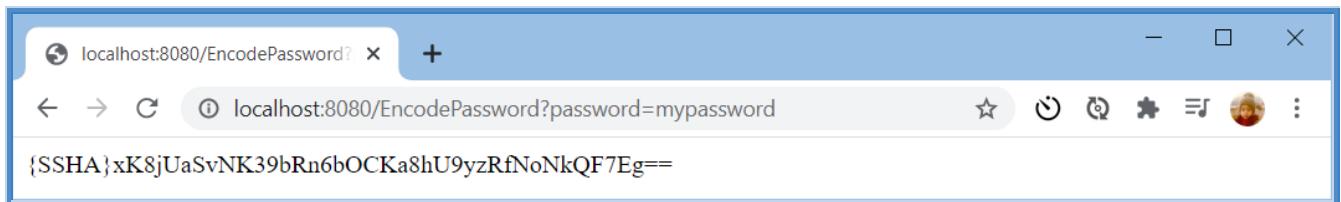


<http://localhost:8080>Hello>

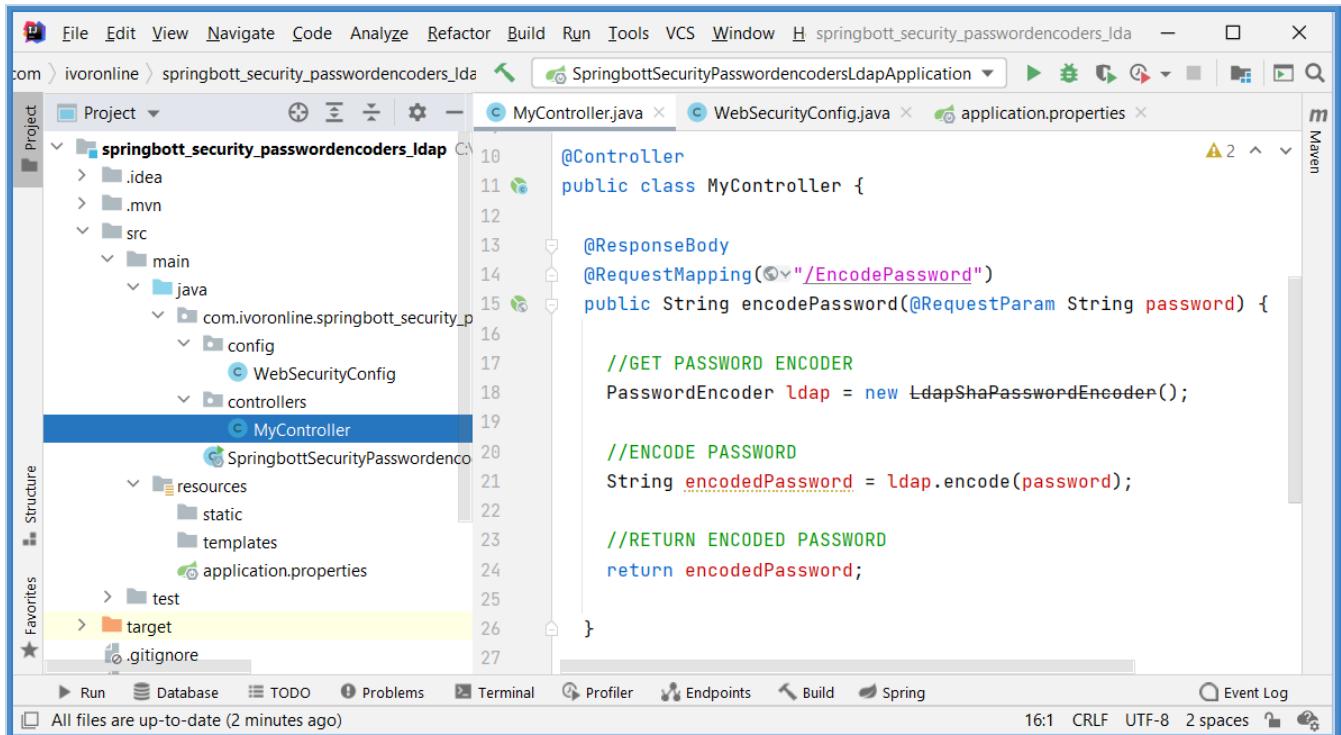


<http://localhost:8080/EncodePassword?password=mypassword>

(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>
```

1.5.2 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)

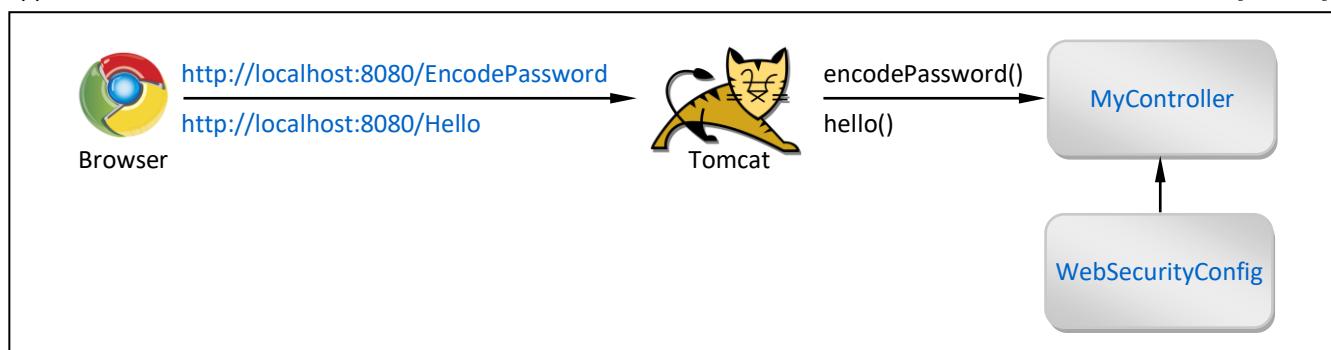
```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-security</artifactId>
</dependency>
```

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 <code>@Controller</code> , <code>@RequestMapping</code> 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}xK8jUaSvNK39bRn6b0CKa8hU9yzRfNoNkQF7Eg==
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.ivorononline.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
    //=====

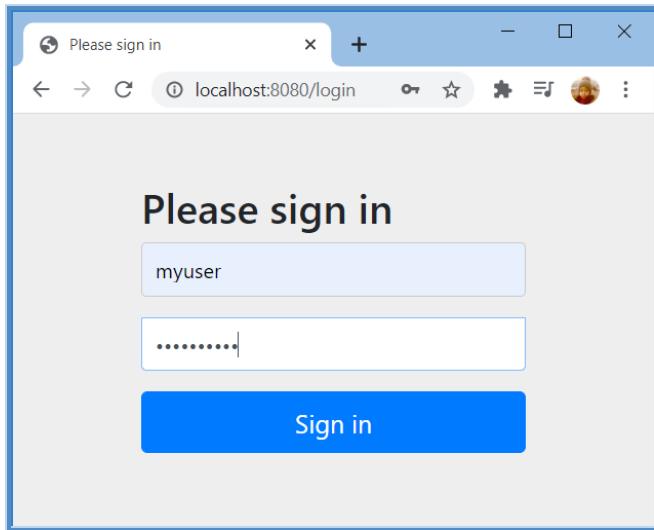
    @Override
    protected void configure(HttpSecurity httpSecurity) throws Exception {
        httpSecurity.authorizeRequests().antMatchers("/EncodePassword").permitAll(); //Anonymous Access
        httpSecurity.authorizeRequests().anyRequest().authenticated(); //Authenticated Access
        httpSecurity.formLogin();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

    }
}
```

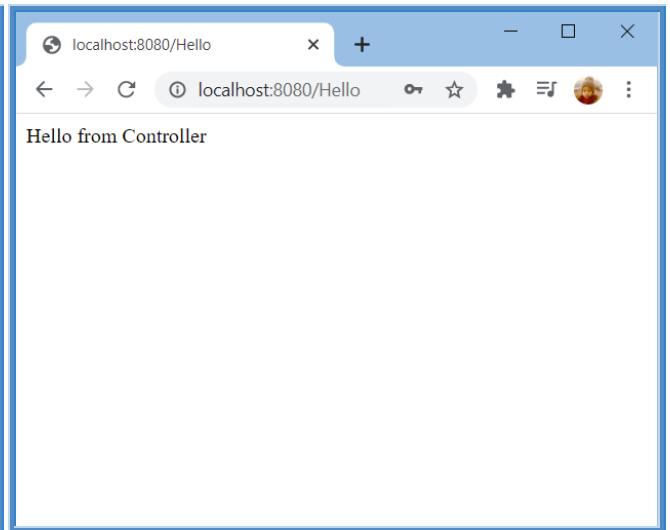
Results

- <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>

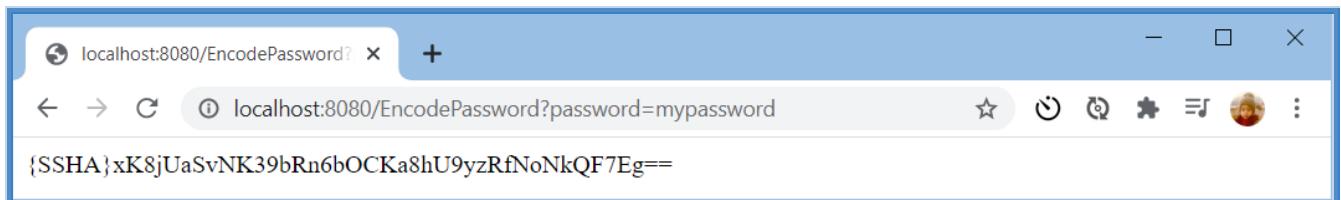


<http://localhost:8080>Hello>

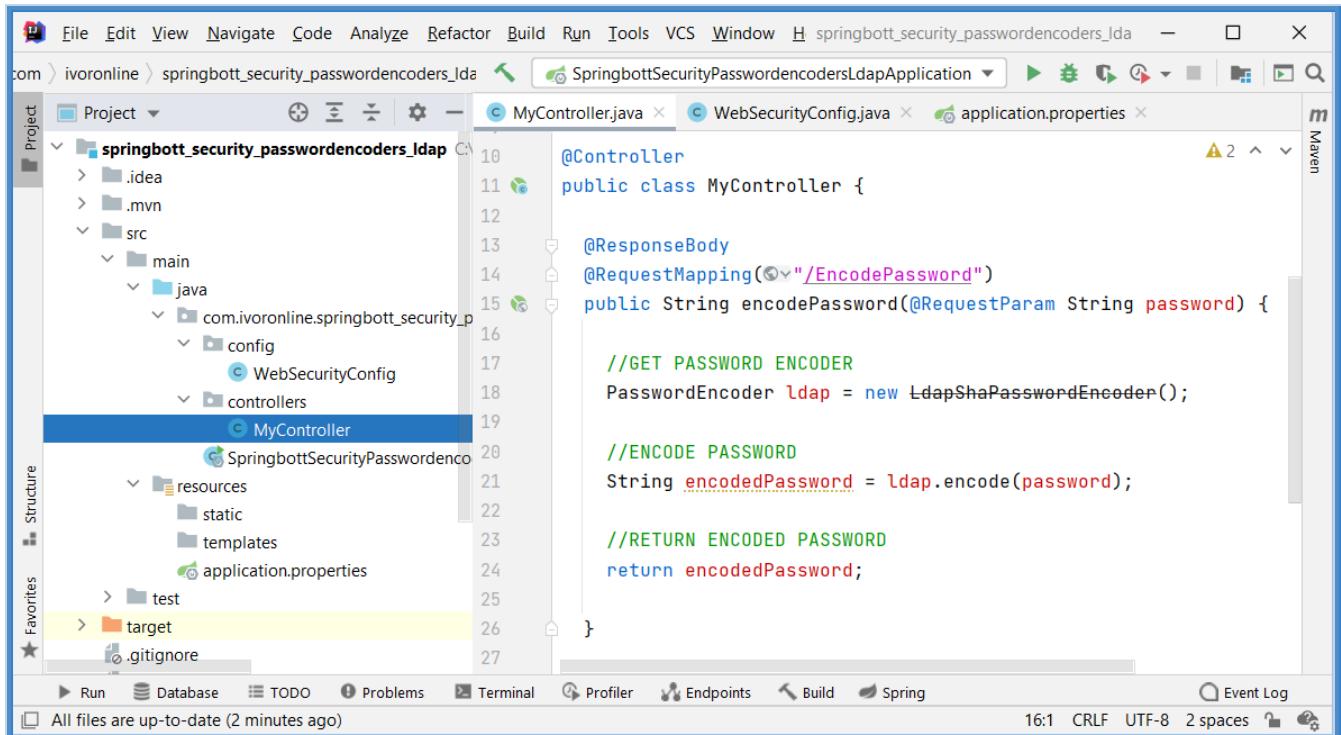


<http://localhost:8080/EncodePassword?password=mypassword>

(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>
```

1.5.3 Sha256

Info

[G]

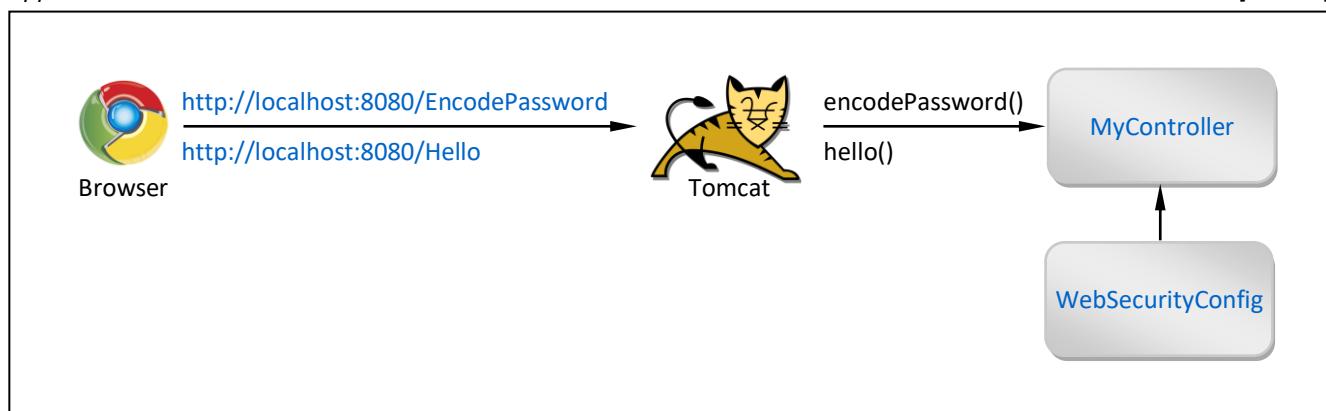
- Sha256 Password Encoder uses **random salt** so that encoded password is always different.
It has `matches()` Method to compare encoded password with the original password.
You can also provide a secret that needs to be applied every time.

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: `springboot_security_passwordencoders_sha256` (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 = 6897d478555b5baf32b35f0f1ffc7d01b9aa77df56458caababca855541f77c9e7265813959758cf
spring.security.user.roles    = USER
```

MyController.java

```
package com.ivorononline.springboot_security_passwordencoders_sha256.controllers;

import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.security.crypto.password.StandardPasswordEncoder;
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 StandardPasswordEncoder();

        //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.springboot_security_passwordencoders_sha256.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.PasswordEncoder;
import org.springframework.security.crypto.password.StandardPasswordEncoder;

@Configuration
@EnableWebSecurity
public class WebSecurityConfig extends WebSecurityConfigurerAdapter {

    //=====
    // PASSWORD ENCODER
    //=====

    @Bean
    PasswordEncoder passwordEncoder() {
        return new StandardPasswordEncoder();
    }

    //=====
    // CONFIGURE
    //=====

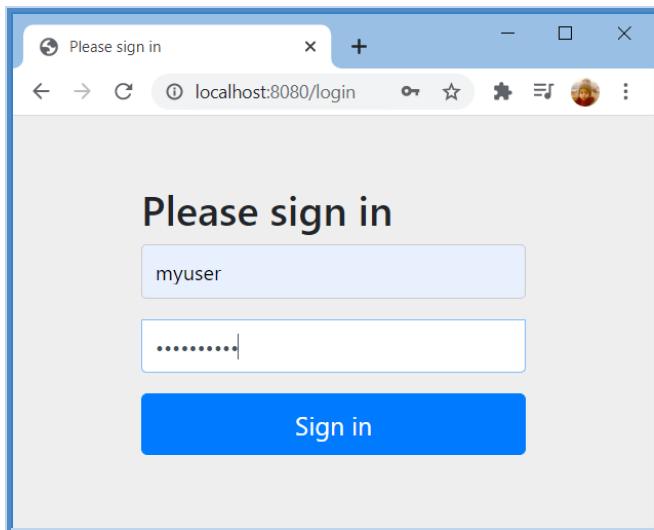
    @Override
    protected void configure(HttpSecurity httpSecurity) throws Exception {
        httpSecurity.authorizeRequests().antMatchers("/EncodePassword").permitAll(); //Anonymous Access
        httpSecurity.authorizeRequests().anyRequest().authenticated(); //Authenticated Access
        httpSecurity.formLogin();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

    }
}
```

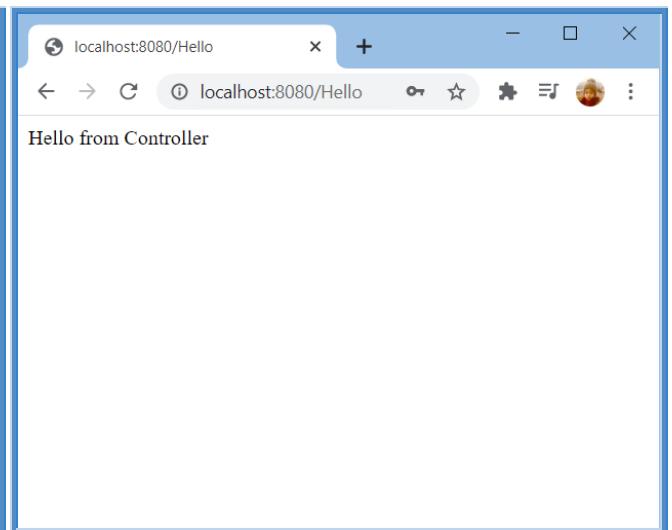
Results

- <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>

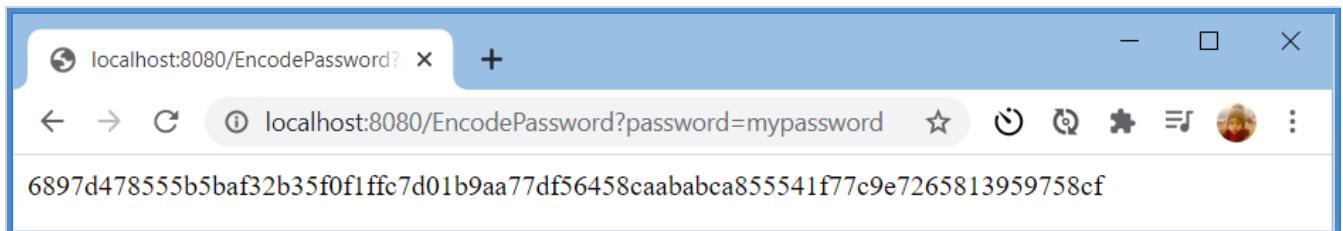


<http://localhost:8080>Hello>

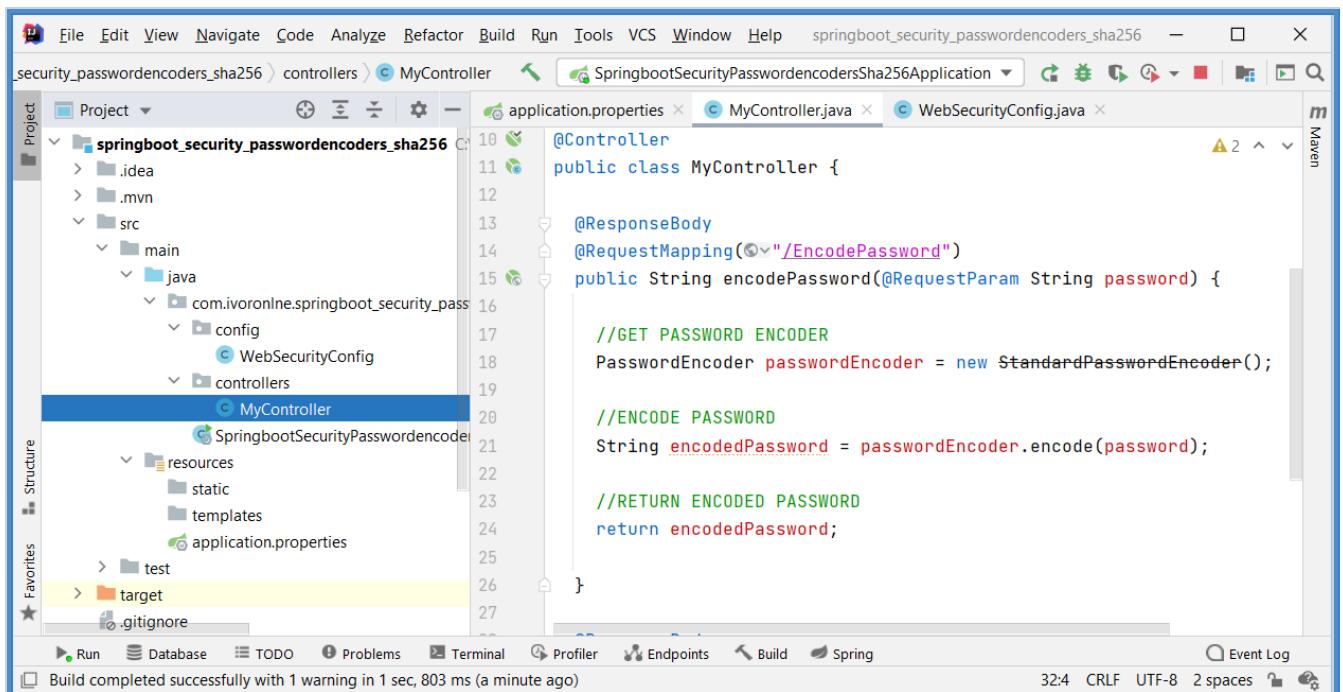


<http://localhost:8080/EncodePassword?password=mypassword>

(if you want to encode another Password)



Application Structure



```
<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>
```

1.5.4 BCrypt

Info

[G]

- BCrypt Password Encoder uses **random salt** so that encoded password is always different.
- It has `matches()` Method to compare encoded password with the original password.
- When instantiating Encoder you can define its Strength. Default value is 10. With bigger value it takes longer to encode.
 - Used Strength is embedded in the result `$2a$10$lsrdF/M4sIGjPOO8G/7cB.w6/SWqkyXCxqZJEZWjObavbMoJVY8QG`.
 - Strength parameter serves as defense against brute force attack so that less combinations can be covered in the same amount of time.

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 `$2a$10$zV89sUNg2HOQn6AQDT.SIO0CzE9W/ZHU095k49pHwJpbckDupYFxG`.

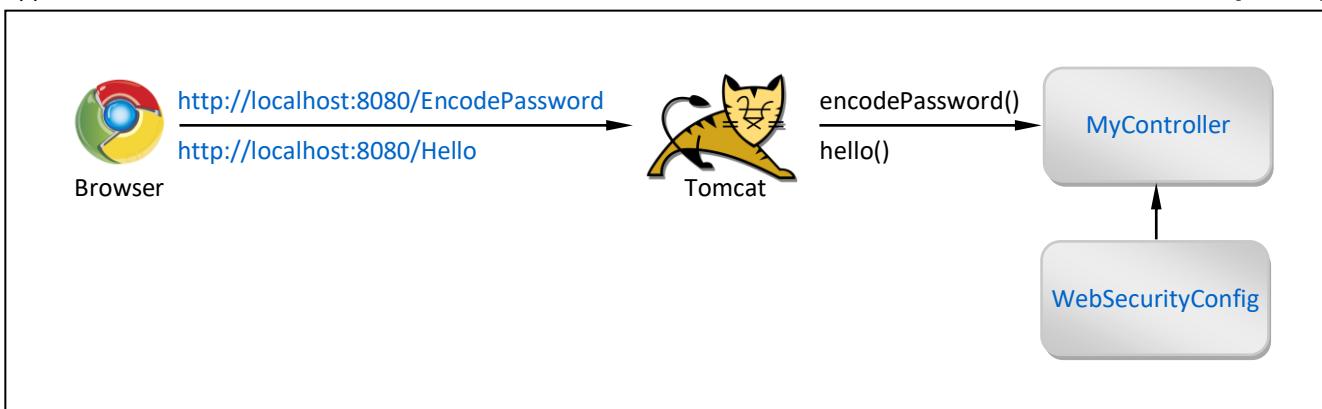
- Inside the Controller we have added `"/EncodePassword"` Endpoint which you can use to encode other Passwords.
Inside `WebSecurityConfig.java` we have allowed Anonymous Access to this Endpoint.

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 <code>@Controller</code> , <code>@RequestMapping</code> and Tomcat Server
Security	Spring Security	Enables Spring Security

Procedure

- Create Project: springbott_security_passwordencoders_bcrypt (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 = $2a$10$zV89sUNg2HOQn6AQDT.SI00CzE9W/ZHU095k49pHwJpbckDupYFxG
spring.security.user.roles     = USER
```

MyController.java

```
package com.ivornline.springboot_passwordencoders_bcrypt.controllers;

import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
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 {

    //=====
    // ENCODE PASSWORD
    //=====

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

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

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

        //RETURN ENCODED PASSWORD
        return encodedPassword;
    }

    //=====
    // HELLO
    //=====

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

WebSecurityConfig.java

```
package com.ivornline.springboot_security_passwordencoders_bcrypt.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.bcrypt.BCryptPasswordEncoder;
import org.springframework.security.crypto.password.PasswordEncoder;

@Configuration
@EnableWebSecurity
public class WebSecurityConfig extends WebSecurityConfigurerAdapter {

    //=====
    // PASSWORD ENCODER
    //=====

    @Bean
    PasswordEncoder passwordEncoder() {
        return new BCryptPasswordEncoder();
    }

    //=====
    // CONFIGURE
    //=====

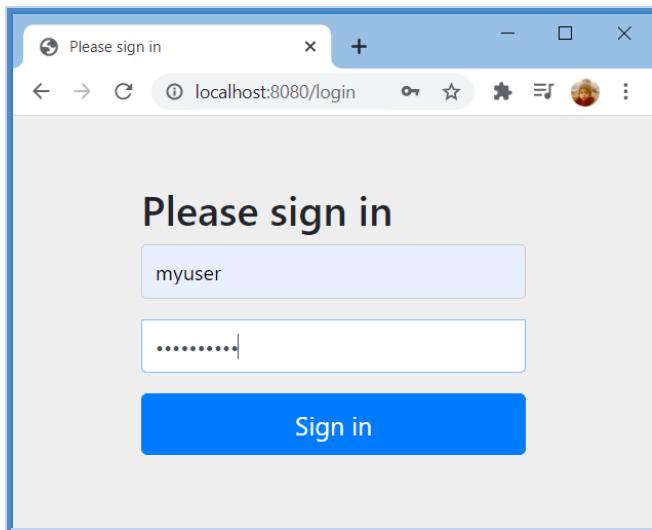
    @Override
    protected void configure(HttpSecurity httpSecurity) throws Exception {
        httpSecurity.authorizeRequests().antMatchers("/EncodePassword").permitAll(); //Anonymous Access
        httpSecurity.authorizeRequests().anyRequest().authenticated(); //Authenticated Access
        httpSecurity.formLogin();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

    }
}
```

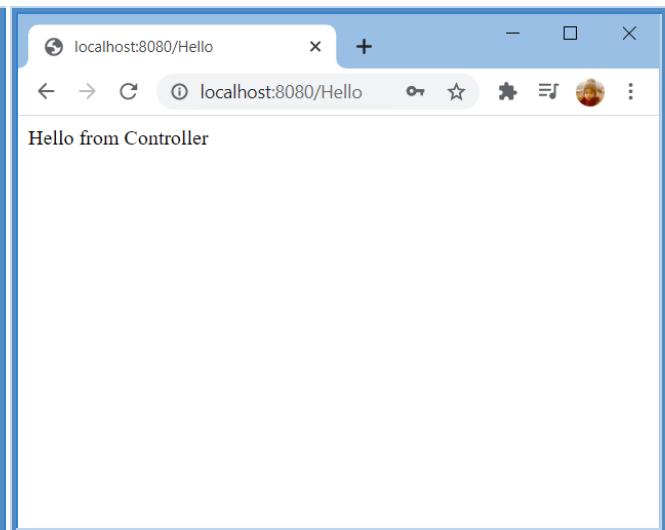
Results

- <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>

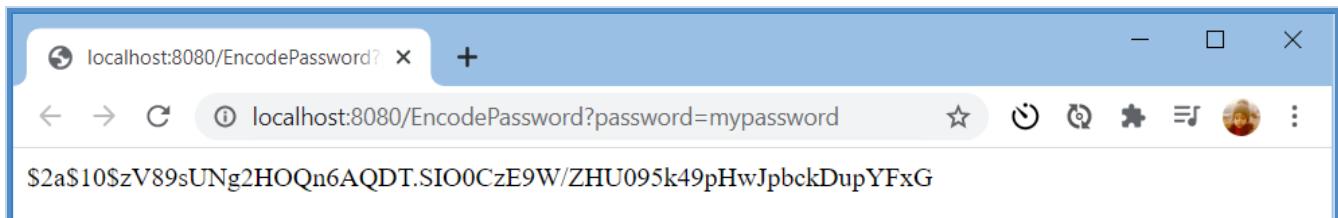


<http://localhost:8080>Hello>

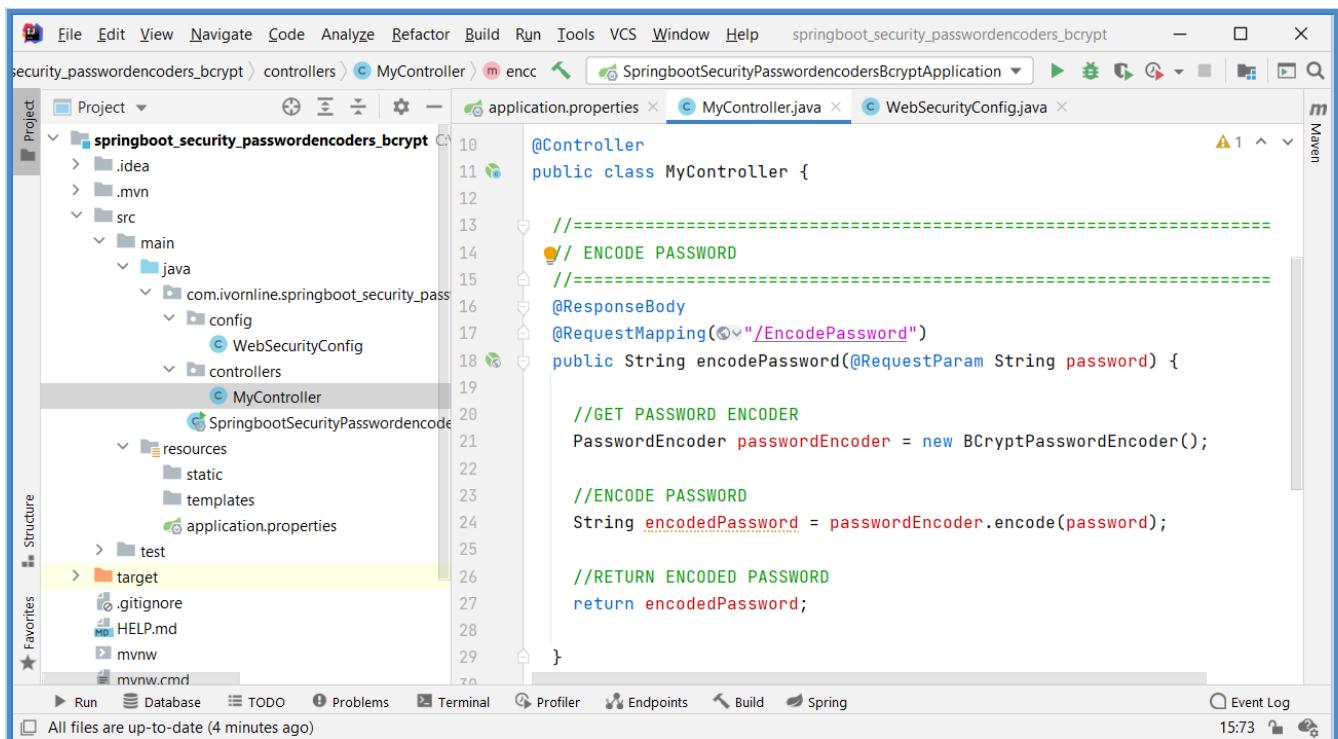


<http://localhost:8080/EncodePassword?password=mypassword>

(if you want to encode another Password)



Application Structure



```
File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help springboot_security_passwordencoders_bcrypt
```

```
security_passwordencoders_bcrypt > controllers > MyController > encode > application.properties > MyController.java > WebSecurityConfig.java
```

```
Project > Project > src > main > java > com.ivorline.springboot_security_pass > config > WebSecurityConfig > controllers > MyController > SpringbootSecurityPasswordencoder > resources > static > templates > application.properties
```

```
application.properties
```

```
MyController.java
```

```
WebSecurityConfig.java
```

```
MyController.java
```

```
10 @Controller
11 public class MyController {
12
13     //=====
14     // ENCODE PASSWORD
15     //=====
16     @ResponseBody
17     @RequestMapping("/EncodePassword")
18     public String encodePassword(@RequestParam String password) {
19
20         //GET PASSWORD ENCODER
21         PasswordEncoder passwordEncoder = new BCryptPasswordEncoder();
22
23         //ENCODE PASSWORD
24         String encodedPassword = passwordEncoder.encode(password);
25
26         //RETURN ENCODED PASSWORD
27         return encodedPassword;
28     }
29 }
```

```
SpringbootSecurityPasswordencodersBcryptApplication
```

```
mvnw.cmd
```

```
mvnw
```

```
mvnw.cmd
```

```
All files are up-to-date (4 minutes ago)
```

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>
```

1.6 Manual Authentication

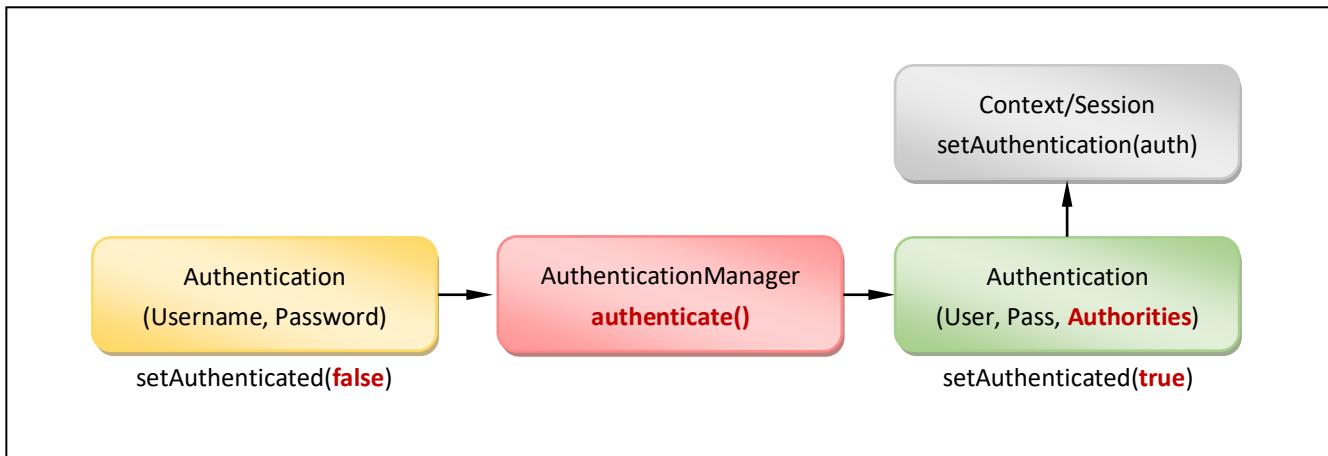
Info

- **Manual Authentication** simply means to **store Authentication inside Context**
 - create an Instance of a Class that Implements Authentication Interface (like UsernamePasswordAuthenticationToken)
 - call `setAuthenticated(true)` on it
 - store Authorities into it
 - store that Instance into Context (by calling `setAuthentication(returnedAuth)`)
 - For each **subsequent HTTP Request** Spring
 - will get Authentication Object from the Context
 - check that `isAuthenticated() == true`
 - get Authorities from it
 - use those Authorities to control access to Controller Endpoints
 - **Session Based Authentication** is used when Application **uses Session**. Then
 - Authentication is stored into Context only once
 - and for each subsequent HTTP Request Spring can get Authentication automatically from the Context/Session
 - **Filter Based Authentication** is used when Application **doesn't use Session**. Then
 - Authentication is stored into Context for each HTTP Request
 - inside Filter
 - which gets executed before Spring gets to the Controller
 - so that when Spring gets to the Controller it can get Authentication from the Context
 - and use Authorities from it to control access to Controller Endpoints

Implementing Manual Authentication

- Both Session and Filter Based Manual Authentication is implemented in the same way.
The only difference is from where the code is called and how many times.
 - We will **implement Manual Authentication** inside `authenticate()` Method of `AuthenticationManager` Interface
For that purpose we will create Class `MyAuthenticationManager` that implements `AuthenticationManager` Interface.
 - We will use two instances of `UsernamePasswordAuthenticationToken` Class that Implements `Authentication` Interface
 - first as Input Parameter for `authenticate()` Method - when it will only contain entered Username and Password
 - secondly as Return Value of `authenticate()` Method - when it will also contain Authorities & `setAuthenticated(true)`

Implementing Manual Authentication



Inside Controller or Filter

```
@Autowired MyAuthenticationManager myAuthenticationManager;

//AUTHENTICATE
Authentication enteredAuth = new UsernamePasswordAuthenticationToken(enteredUsername, enteredPassword);
Authentication returnedAuth = myAuthenticationManager.authenticate(enteredAuth);

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

MyAuthenticationManager.java

```
@Component
public class MyAuthenticationManager implements AuthenticationManager {

    @Override
    public Authentication authenticate(Authentication enteredAuthentication) {

        //HARD CODED USER
        String username = "myuser";
        String password = "mypassword";
        String role      = "ROLE_USER";

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

        //AUTHENTICATE USER
        if (!enteredUsername.equals(username)) { System.out.println("Username not found"); return null; }
        if (!enteredPassword.equals(password)) { System.out.println("Incorrect Password"); return null; }

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

        //CREATE VALIDATED AUTHENTICATION
        Authentication validatedAuthentication = new
        UsernamePasswordAuthenticationToken(username, password, authorities);

        //RETURN VALIDATES AUTHENTICATION
        return validatedAuthentication;

    }

}
```

1.6.1 Theory - Classes & Interfaces

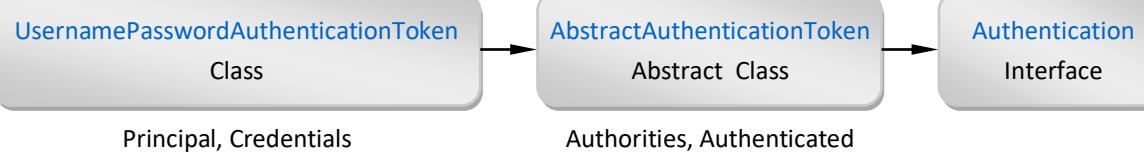
Info

- This tutorial covers more theory about Classes & Interfaces that are involved in [Implementing Manual Authentication](#).

Content

AuthenticationManager Interface	Authentication authenticate(Authentication authentication)
Authentication - Interface	Principal, Credentials, Authorities, Authenticated
AbstractAuthenticationToken - Abstract Class	Authorities, Authenticated
UsernamePasswordAuthenticationToken - Class	Principal, Credentials

Hierarchy



AuthenticationManager Interface

- To manually Authenticate User
 - declare Class that implements **AuthenticationManager Interface**
 - @Override authenticate() Method**
- authenticate()** Method accepts and returns instance of Class that implements **Authentication Interface**.

AuthenticationManager.java

```
public interface AuthenticationManager {  
    Authentication authenticate(Authentication authentication) throws AuthenticationException;  
}
```

Authentication Interface

- Authentication Interface specifies below Methods.

Authentication.java

```
public interface Authentication extends Principal, Serializable {  
  
    Object getPrincipal();  
    Object getCredentials();  
  
    Collection<? extends GrantedAuthority> getAuthorities();  
  
    boolean isAuthenticated();  
    void setAuthenticated(boolean var1) throws IllegalArgumentException;  
  
    Object getDetails();  
  
}
```

AbstractAuthenticationToken Abstract Class

- Abstract Class `AbstractAuthenticationToken` partially implements `Authentication` Interface by providing
 - Properties: `authorities`, `authenticated`
 - Methods: `getAuthorities()`, `isAuthenticated()`, `setAuthenticated()`
- Remaining Properties & Methods are implemented by `UsernamePasswordAuthenticationToken`
 - Properties: `principal`, `credentials`
 - Methods: `getPrincipal()`, `getCredentials()`

AbstractAuthenticationToken.java

```
public abstract class AbstractAuthenticationToken implements Authentication, CredentialsContainer {  
  
    //PROPERTIES  
    private boolean authenticated = false;  
    private final Collection<GrantedAuthority> authorities;  
  
    //METHODS FOR AUTHENTICATED & AUTHORITIES  
    public boolean isAuthenticated () { return this.authenticated; }  
    public void setAuthenticated(boolean authenticated) { this.authenticated = authenticated; }  
  
    public Collection<GrantedAuthority> getAuthorities() { return this.authorities; }  
  
    ...  
}
```

UsernamePasswordAuthenticationToken Class

- **UsernamePasswordAuthenticationToken** extends **AbstractAuthenticationToken** and implements remaining parts of **Authentication Interface**, therefore it can be used as Input Parameter and as Return Value for **authenticate()** Method.
- We will create two instances of this Class and use it like this
 - First we create an instance by calling Constructor that only takes **entered** Username and Password
This Constructor calls **setAuthenticated(false)**.
This instance is then used as **Input Parameter** to **authenticate()** Method.
 - If validation was successful then inside **authenticate()** Method we create another instance by calling Constructor that takes Authorities as additional third Parameter.
This Constructor calls **setAuthenticated(true)**.
This instance is then **returned** from **authenticate()** Method.
- **UsernamePasswordAuthenticationToken** has **setAuthenticated()** but you are not allowed to call it directly with **true**.
This is shown in the below code (highlighted in red) which states that you have to call second Constructor instead (the one that takes Authorities as third parameter) and which calls **super.setAuthenticated(true)**.

UsernamePasswordAuthenticationToken.java

```
public class UsernamePasswordAuthenticationToken extends AbstractAuthenticationToken {  
  
    //PROPERTIES  
    private final Object principal;  
    private Object credentials;  
  
    //METHODS FOR PRINCIPAL & CREDENTIALS  
    public UsernamePasswordAuthenticationToken(Object principal, Object credentials) {  
        super((Collection)null);  
        this.principal = principal;  
        this.credentials = credentials;  
        this.setAuthenticated(false);  
    }  
  
    public UsernamePasswordAuthenticationToken(Object principal, Object credentials, Collection<? extends GrantedAuthority> authorities) {  
        super(authorities);  
        this.principal = principal;  
        this.credentials = credentials;  
        super.setAuthenticated(true);  
    }  
  
    public Object getCredentials() { return this.credentials; }  
  
    public Object getPrincipal() { return this.principal; }  
  
    public void setAuthenticated(boolean isAuthenticated) throws IllegalArgumentException {  
        Assert.isTrue(!isAuthenticated, "Cannot set this token to trusted - use constructor which takes a GrantedAuthority list instead");  
        super.setAuthenticated(false);  
    }  
}
```

1.6.2 Single Time (Session Based) - Request Parameters

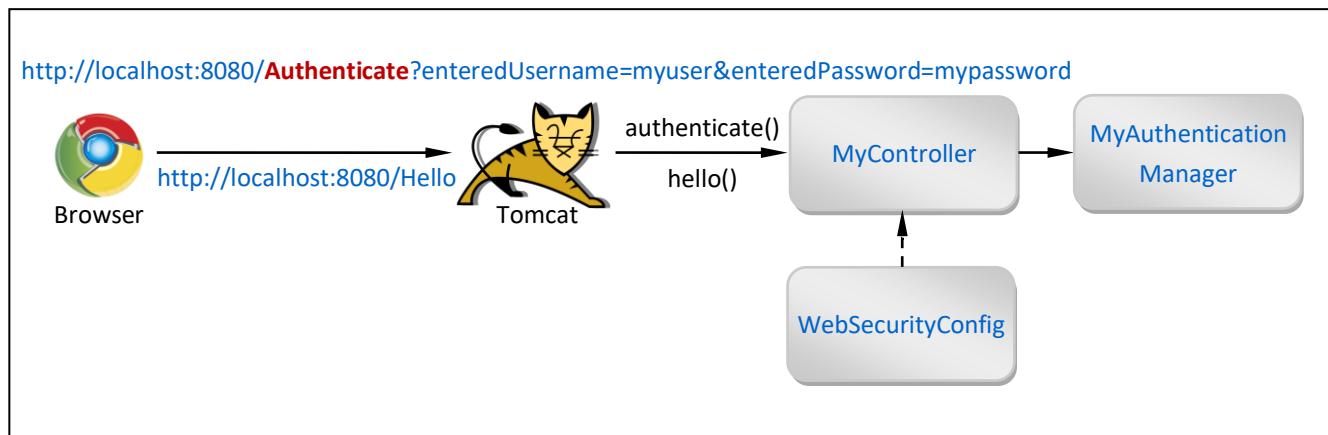
Info

[G]

- This tutorial shows how to use **Session to Implement Manual Authentication** by
 - calling <http://localhost:8080/Authenticate?enteredUsername=myuser&enteredPassword=mypassword>
 - providing Credentials as HTTP Request Parameters
 - storing Authentication Object into Context/Session
 - from where it will be used/accessed for every subsequent HTTP Request

Application Schema

[Results]



Spring Boot Starters

GROUP	DEPENDENCY	DESCRIPTION
Web	Spring Web	Enables Controller Annotations and Tomcat Server
Security	Spring Security	Enables Spring Security

Procedure

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

MyAuthenticationManager.java

```
package com.ivoronline.springboot_authentication_manual.config;

import org.springframework.security.authentication.AuthenticationManager;
import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;
import org.springframework.security.core.Authentication;
import org.springframework.security.core.GrantedAuthority;
import org.springframework.security.core.authority.SimpleGrantedAuthority;
import org.springframework.stereotype.Component;
import java.util.ArrayList;
import java.util.List;

@Component
public class MyAuthenticationManager implements AuthenticationManager {

    @Override
    public Authentication authenticate(Authentication enteredAuthentication) {

        //HARD CODED USER
        String username = "myuser";
        String password = "mypassword";
        String role     = "ROLE_USER";

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

        //AUTHENTICATE USER
        if (!enteredUsername.equals(username)) { System.out.println("Username not found"); return null; }
        if (!enteredPassword.equals(password)) { System.out.println("Incorrect Password"); return null; }

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

        //CREATE VALIDATED AUTHENTICATION
        Authentication validatedAuth = new UsernamePasswordAuthenticationToken(username,password,authorities);

        //RETURN VALIDATES AUTHENTICATION
        return validatedAuth;

    }
}
```

WebSecurityConfig.java

```
package com.ivoronline.springboot_authentication_manual.config;

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;

@Configuration
@EnableWebSecurity
public class WebSecurityConfig extends WebSecurityConfigurerAdapter {

    //=====
    // CONFIGURE
    //=====

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

        //SPECIFY ACCESS TO ENDPOINTS
        httpSecurity.authorizeRequests().antMatchers("/Authenticate").permitAll(); //Anonymous Access
        httpSecurity.authorizeRequests().anyRequest().authenticated(); //Authenticated Access

    }

}
```

MyController.java

```
package com.ivorononline.springboot_authentication_manual.controllers;

import com.ivorononline.springboot_authentication_manual.config.MyAuthenticationManager;
import org.springframework.beans.factory.annotation.Autowired;
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.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.ResponseBody;

@Controller
public class MyController {

    @Autowired MyAuthenticationManager myAuthenticationManager;

    //=====
    // AUTHENTICATE
    //=====

    @ResponseBody
    @RequestMapping("/Authenticate")
    public String authenticate(@RequestParam String enteredUsername, @RequestParam String enteredPassword) {

        //AUTHENTICATE
        Authentication enteredAuth = new UsernamePasswordAuthenticationToken(enteredUsername, enteredPassword);
        Authentication returnedAuth = myAuthenticationManager.authenticate(enteredAuth);

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

        //RETURN SOMETHING
        if(returnedAuth == null) { return "User is NOT Authenticated"; }
        else { return "User is Authenticated"; }
    }

    //=====
    // HELLO
    //=====

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

Results

<http://localhost:8080>Hello>

(before Authentication)



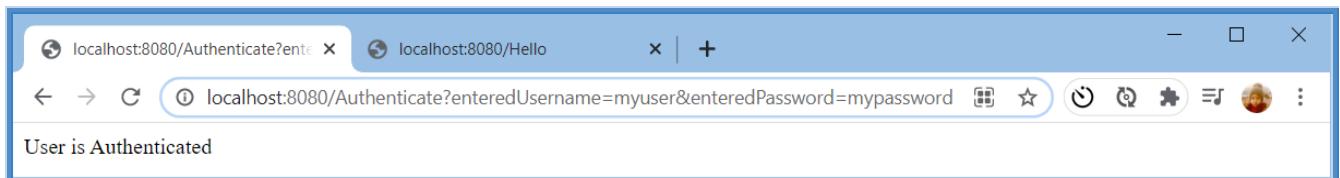
Whitelabel Error Page

This application has no explicit mapping for /error, so you are seeing this as a fallback.

Sat Feb 06 17:13:54 CET 2021

There was an unexpected error (type=Forbidden, status=403).

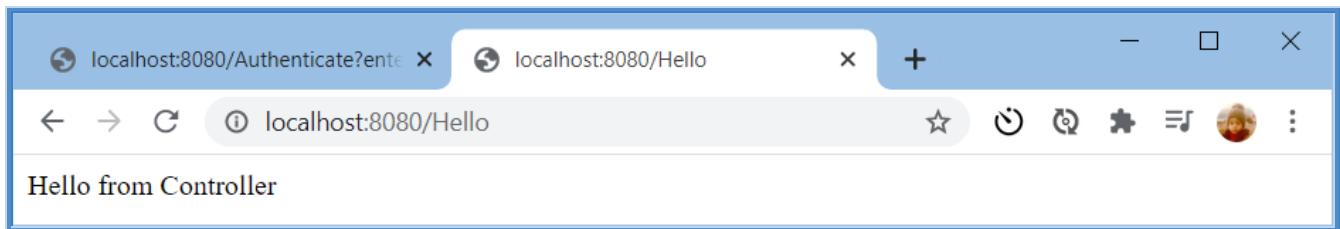
<http://localhost:8080/Authenticate?enteredUsername=myuser&enteredPassword=mypassword>



User is Authenticated

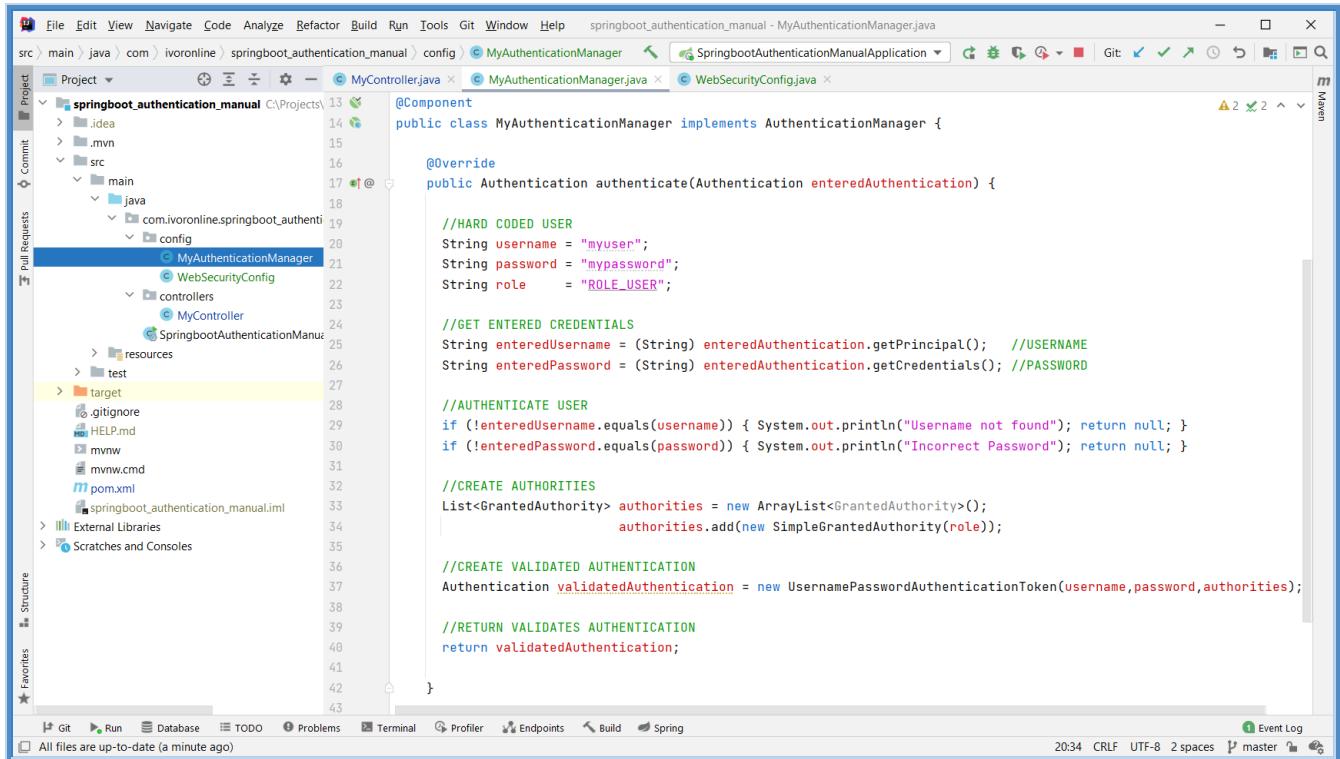
<http://localhost:8080>Hello>

(after Authentication)



Hello from Controller

Application Structure



```
File Edit View Navigate Code Analyze Refactor Build Run Tools Git Window Help springboot_authentication_manual - MyAuthenticationManager.java
src > main > java > com > ivoronline > springboot_authentication_manual > config > MyAuthenticationManager.java > SpringbootAuthenticationManualApplication.java > WebSecurityConfig.java > ...
Project > springboot_authentication_manual > src > main > java > com > ivoronline > springboot_authentication_manual > config > MyAuthenticationManager.java > ...
@Component
public class MyAuthenticationManager implements AuthenticationManager {
    @Override
    public Authentication authenticate(Authentication enteredAuthentication) {
        //HARD CODED USER
        String username = "myuser";
        String password = "mypassword";
        String role     = "ROLE_USER";

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

        //AUTHENTICATE USER
        if (!enteredUsername.equals(username)) { System.out.println("Username not found"); return null; }
        if (!enteredPassword.equals(password)) { System.out.println("Incorrect Password"); return null; }

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

        //CREATE VALIDATED AUTHENTICATION
        Authentication validatedAuthentication = new UsernamePasswordAuthenticationToken(username, password, authorities);

        //RETURN VALIDATES AUTHENTICATION
        return validatedAuthentication;
    }
}
```

```
<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>
```

1.6.3 Every Time (Filter Based) - Request Parameters

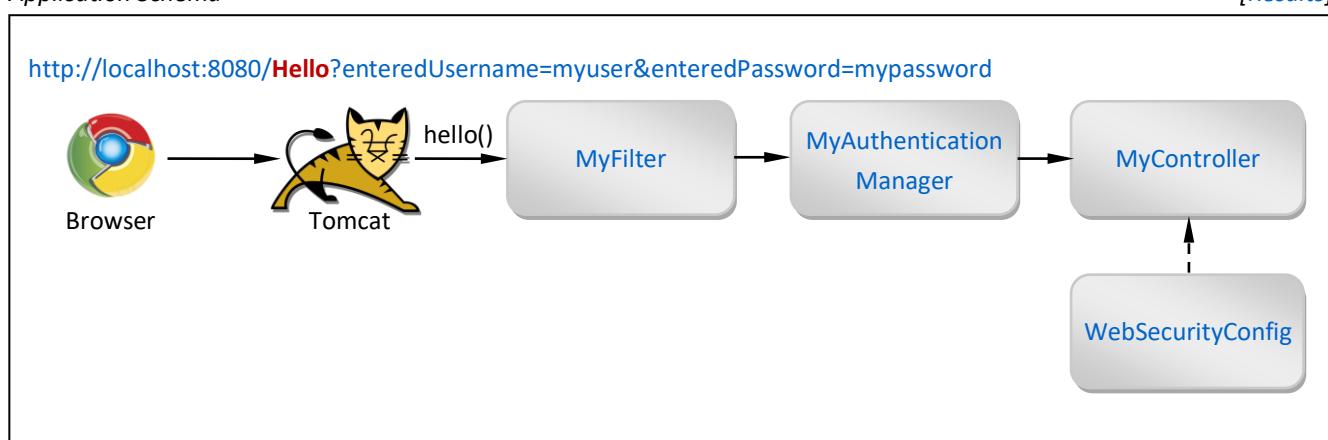
Info

[G]

- This tutorial shows how to use **Filter** to [Implement Manual Authentication](#) by
 - providing Credentials for every HTTP Request
 - as HTTP Request Parameters `?enteredUsername=myuser&enteredPassword=mypassword`
 - so that **Filter** can store Authentication Object into Context/Session for every HTTP Request
 - before HTTP Request is forwarded to the Controller
- Credentials will be compared with the hard coded User inside [MyAuthenticationManager](#).
- This Filter based approach is used when Session is disabled so that no JSESSIONID Cookie is saved in the Browser. Session is disabled in [WebSecurityConfig.java](#) with `sessionCreationPolicy(SessionCreationPolicy.STATELESS)`.

Application Schema

[Results]



Spring Boot Starters

GROUP	DEPENDENCY	DESCRIPTION
Web	Spring Web	Enables Controller Annotations and Tomcat Server
Security	Spring Security	Enables Spring Security

Procedure

- Create Project: `springboot_authentication_manual_filter` (add Spring Boot Starters from the table)
- Create Package: `config` (inside main package)
 - Create Class: `MyAuthenticationManager.java` (inside config 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)

`MyAuthenticationManager.java`

```
package com.ivoronline.springboot_authentication_manual_filter.config;

import org.springframework.security.authentication.AuthenticationManager;
import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;
import org.springframework.security.core.Authentication;
import org.springframework.security.core.GrantedAuthority;
import org.springframework.security.core.authority.SimpleGrantedAuthority;
import org.springframework.stereotype.Component;
import java.util.ArrayList;
import java.util.List;

@Component
public class MyAuthenticationManager implements AuthenticationManager {

    @Override
    public Authentication authenticate(Authentication enteredAuthentication) {

        //HARD CODED USER
        String username = "myuser";
        String password = "mypassword";
        String role      = "ROLE_USER";

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

        //AUTHENTICATE USER
        if (!enteredUsername.equals(username)) { System.out.println("Username not found"); return null; }
        if (!enteredPassword.equals(password)) { System.out.println("Incorrect Password"); return null; }

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

        //CREATE VALIDATED AUTHENTICATION
        Authentication validatedAuth = new UsernamePasswordAuthenticationToken(username,password,authorities);

        //RETURN VALIDATES AUTHENTICATION
        return validatedAuth;

    }

}
```

MyFilter.java

```
package com.ivorononline.springboot_authentication_manual_filter.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 org.springframework.beans.factory.annotation.Autowired;
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
    MyAuthenticationManager myAuthenticationManager;

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

        //GET CREDENTIALS
        String enteredUsername = request.getParameter("enteredUsername");
        String enteredPassword = request.getParameter("enteredPassword");

        //AUTHENTICATE
        Authentication enteredAuth = new UsernamePasswordAuthenticationToken(enteredUsername, enteredPassword);
        Authentication returnedAuth = myAuthenticationManager.authenticate(enteredAuth);

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

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

WebSecurityConfig.java

```
package com.ivoronline.springboot_authentication_manual_filter.config;

import org.springframework.beans.factory.annotation.Autowired;
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.EnableWebSecurity;
import org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;
import org.springframework.security.config.http.SessionCreationPolicy;
import org.springframework.security.web.authentication.UsernamePasswordAuthenticationFilter;

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

    @Autowired private MyFilter myFilter;

    @Override
    protected void configure(HttpSecurity httpSecurity) throws Exception {
        httpSecurity.authorizeRequests().antMatchers("/Authenticate").permitAll();           //Anonymous
        httpSecurity.authorizeRequests().anyRequest().authenticated();                      //Authenticated
        httpSecurity.sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS); //No Session
        httpSecurity.addFilterBefore(myFilter, UsernamePasswordAuthenticationFilter.class);   //Add Filter
    }
}
```

MyController.java

```
package com.ivoronline.springboot_authentication_manual_filter.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 hello() {
        return "Hello from Controller";
    }
}
```

Results

<http://localhost:8080>Hello?enteredUsername=myuser&enteredPassword=mypassword>

(no Session Cookie)

The screenshot shows a browser window with the URL <http://localhost:8080>Hello?enteredUsername=myuser&enteredPassword=mypassword>. The page content is "Hello from Controller". Below the browser is the Network tab of the developer tools, specifically the Application section. It shows a table for Cookies under the domain <http://localhost:8080>. The table has columns for Name, Value, Domain, Path, Expires /..., Session, HttpOnly, Secure, SameSite, and Priority. There are no rows in the table.

Application Structure

The screenshot shows an IDE (IntelliJ IDEA) displaying a Java application named "springboot_authentication_manual_filter". The Project tool window on the left shows the directory structure: .idea, .mvn, src/main/java/com/ivoronline/springboot_authentication_manual_filter/config, and src/main/java/com/ivoronline/springboot_authentication_manual_filter/config/MyFilter.java. The code editor shows the MyFilter.java file:

```
@Component
public class MyFilter implements Filter {
    @Autowired
    public void doFilter(ServletRequest request, ServletResponse response, FilterChain filterchain)
        throws IOException, ServletException {
        //GET CREDENTIALS
        String enteredUsername = request.getParameter("enteredUsername");
        String enteredPassword = request.getParameter("enteredPassword");

        //AUTHENTICATE
        Authentication enteredAuth = new UsernamePasswordAuthenticationToken(enteredUsername, enteredPassword);
        Authentication returnedAuth = myAuthenticationManager.authenticate(enteredAuth);

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

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

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>
```

1.6.4 Every Time (Filter Based) - Request Headers

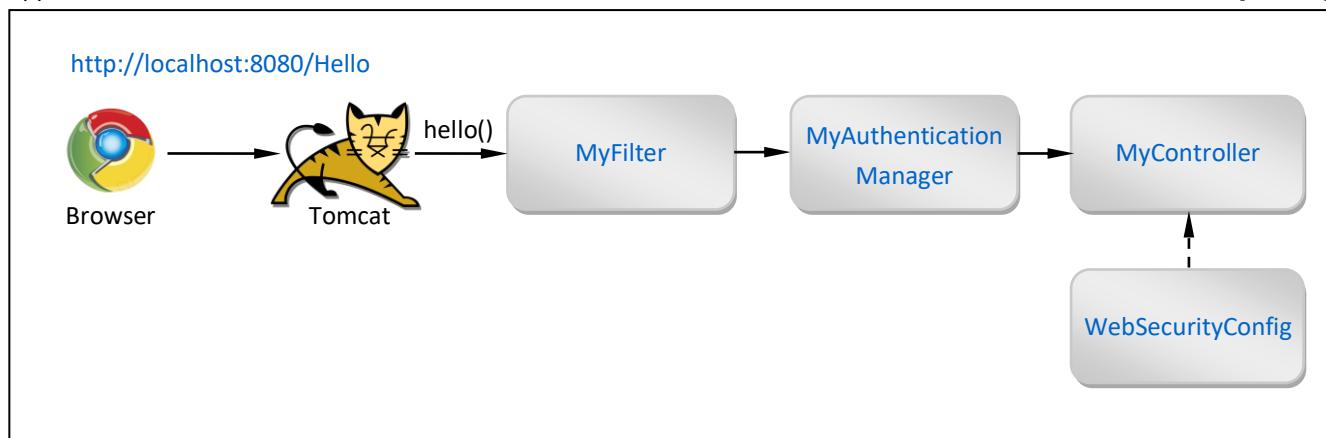
Info

[G]

- This tutorial shows how to use **Filter** to [Implement Manual Authentication](#) by
 - providing Credentials for every HTTP Request
 - in Request Headers `enteredUsername` and `enteredPassword`
 - so that **Filter** can store Authentication Object into Context/Session for every HTTP Request
 - before HTTP Request is forwarded to the Controller
 - Credentials will be compared with the hard coded User inside [MyAuthenticationManager](#).
 - This Filter based approach is used when Session is disabled so that no JSESSIONID Cookie is saved in the Browser. Session is disabled in [WebSecurityConfig.java](#) with `sessionCreationPolicy(SessionCreationPolicy.STATELESS)`. We also need to disable CSRF for POST Request to work `httpSecurity.csrf().disable()`.
- Changes compared to [Every Time \(Filter Based\) - Request Parameters](#) tutorial are highlighted in blue.

Application Schema

[Results]



Spring Boot Starters

GROUP	DEPENDENCY	DESCRIPTION
Web	Spring Web	Enables Controller Annotations and Tomcat Server
Security	Spring Security	Enables Spring Security

Procedure

- **Create Project:** `springboot_authentication_manual_filter` (add Spring Boot Starters from the table)
- **Create Package:** `config` (inside main package)
 - **Create Class:** `MyAuthenticationManager.java` (inside config 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)

MyAuthenticationManager.java

```
package com.ivoronline.springboot_authentication_manual_filter_headers.config;

import org.springframework.security.authentication.AuthenticationManager;
import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;
import org.springframework.security.core.Authentication;
import org.springframework.security.core.GrantedAuthority;
import org.springframework.security.core.authority.SimpleGrantedAuthority;
import org.springframework.stereotype.Component;
import java.util.ArrayList;
import java.util.List;

@Component
public class MyAuthenticationManager implements AuthenticationManager {

    @Override
    public Authentication authenticate(Authentication enteredAuthentication) {

        //HARD CODED USER
        String username = "myuser";
        String password = "mypassword";
        String role      = "ROLE_USER";

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

        //AUTHENTICATE USER
        if (!enteredUsername.equals(username)) { System.out.println("Username not found"); return null; }
        if (!enteredPassword.equals(password)) { System.out.println("Incorrect Password"); return null; }

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

        //CREATE VALIDATED AUTHENTICATION
        Authentication validatedAuth = new UsernamePasswordAuthenticationToken(username,password,authorities);

        //RETURN VALIDATES AUTHENTICATION
        return validatedAuth;

    }

}
```

MyFilter.java

```
package com.ivorononline.springboot_authentication_manual_filter_headers.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.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

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

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

        //CAST TO GET ACCESS TO HEADERS
        HttpServletRequest httpRequest = (HttpServletRequest) request;

        //GET CREDENTIALS
        String enteredUsername = httpRequest.getHeader("enteredUsername");
        String enteredPassword = httpRequest.getHeader("enteredPassword");

        //AUTHENTICATE
        Authentication enteredAuth = new UsernamePasswordAuthenticationToken(enteredUsername, enteredPassword);
        Authentication returnedAuth = myAuthenticationManager.authenticate(enteredAuth);

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

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

WebSecurityConfig.java

```
package com.ivoronline.springboot_authentication_manual_filter_headers.config;

import org.springframework.beans.factory.annotation.Autowired;
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.EnableWebSecurity;
import org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;
import org.springframework.security.config.http.SessionCreationPolicy;
import org.springframework.security.web.authentication.UsernamePasswordAuthenticationFilter;

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

    @Autowired private MyFilter myFilter;

    @Override
    protected void configure(HttpSecurity httpSecurity) throws Exception {
        httpSecurity.authorizeRequests().antMatchers("/Authenticate").permitAll();           //Anonymous
        httpSecurity.authorizeRequests().anyRequest().authenticated();                     //Authenticated
        httpSecurity.sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS); //No Session
        httpSecurity.addFilterBefore(myFilter, UsernamePasswordAuthenticationFilter.class);   //Add Filter
        httpSecurity.csrf().disable();                                                     //Enables POST
    }

}
```

MyController.java

```
package com.ivoronline.springboot_authentication_manual_filter_headers.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 hello() {
        return "Hello from Controller";
    }
}
```

Results

- Start Postman
- POST: <http://localhost:8080>Hello>
- Headers: (copy from below)
- Send

Headers

(add Key-Value)

```
enteredUsername: myuser  
enteredPassword: mypassword
```

POST <http://localhost:8080>Hello>

The screenshot shows the Postman interface with a POST request to <http://localhost:8080>Hello>. The Headers tab is active, showing the following configuration:

Key	Description
Content-Type	application/json
enteredUsername	myuser
enteredPassword	mypassword

The status bar at the bottom indicates a 200 OK response with a time of 75 ms and a size of 367 B.

Application Structure

The screenshot shows the IntelliJ IDEA interface with the project `springboot_authentication_manual_filter_headers` open. The `MyFilter.java` file is the active editor, displaying the following code:

```
@Component  
public class MyFilter implements Filter {  
  
    @Autowired  
    public void doFilter(ServletRequest request, ServletResponse response, FilterChain filterchain)  
        throws IOException, ServletException {  
  
        //CAST TO GET ACCESS TO HEADERS  
        HttpServletRequest httpRequest = (HttpServletRequest) request;  
  
        //GET CREDENTIALS  
        String enteredUsername = httpRequest.getHeader("enteredUsername");  
        String enteredPassword = httpRequest.getHeader("enteredPassword");  
  
        //AUTHENTICATE  
        Authentication enteredAuth = new UsernamePasswordAuthenticationToken(enteredUsername, enteredPassword);  
        Authentication returnedAuth = myAuthenticationManager.authenticate(enteredAuth);  
  
        //STORE AUTHENTICATION INTO CONTEXT (SESSION)  
        SecurityContextHolder.getContext().setAuthentication(returnedAuth);  
  
        //FORWARD REQUEST  
        filterchain.doFilter(request, response);  
    }  
}
```

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>
```

1.7 CSRF (Cross-Site Request Forgery)

Info

[R]

- ➊ Cross-Site Request Forgery (CSRF) is an attack that
 - executes HTTP Request (without User's knowledge)
 - from User's Browser (that has Session Token to automatically Authenticates User)
 - to Web Application (in which User is currently authenticated)
- ➋ Such HTTP Request will then contain Session Token that automatically Authenticates User.
Session Token is sent because HTTP Request come from User's Browser (that has Session Token).
Web Application will then execute command contained inside the HTTP Request because it knows that HTTP Request came from already Authenticated User/Browser.
But Web Application doesn't know that User didn't want to execute that command.
- ➌ [CSRF Token](#) is a way of preventing CSRF Attack.
CSRF Token is a random value sent by Web App which is stored in a hidden Field inside the Form.
It is resent to the Web App when user submits the Form in order to Authenticates Web Page from which Request came.

1.7.1 Theory - Normal User Interaction

Info

- User sends initial/first HTTP Request to a Web Application.
- Web Application returns **Login Form** to **Authenticate** the User.
- User must enter his
 - **Username** to tell the Web Application who he is
 - **Password** to prove to the Web Application that he really is who he is claiming to be
- If User is properly Authenticated (Password matches the provided Username) Web Application sends HTTP Response
 - informing User that he was able to log in (or simply forwarding User to the Page where User can make further actions)
 - that includes **Session Cookie** that is stored locally on User's Browser to identify him during subsequent HTTP Requests
- After that every time a User sends a HTTP Request to this Web Application, Session Cookie created by that Web Application is also sent with the HTTP Request to automatically Authenticate User into Web Application.
- At some point, while using Gmail, User might decide to change the Password.
After entering new Password and pressing Submit following HTTP Request will be sent to Gmail to change the Password. Gmail will change the Password because HTTP Request will contain **Session Cookie** to automatically Authenticate User.

HTTP Request to change Password

```
https://www.gmail.com/changepassword?newpassword=mynewpassword
```

Authenticate

- Authentication answers the question: Who are you?
- Server needs to know who you are so that it can only allow you to do things that you are allowed to do.
- For instance if you are logging into Gmail you are only allowed to see your own Emails.
- Entering just Username is not enough because you might be lying that you are really the person behind that Username.
- That is because Username (like Email) are public - other people know your Username.
- So you also need to provide your Password to that you are really the person behind that Username.
- This is because the Password is private - only the person that really owns this Username would now its Password.

Login Form @ <http://www.gmail.com/login>

The image shows a simplified representation of a login interface. At the top center, the text "Please sign in" is displayed. Below it are two input fields: one labeled "Username" and another labeled "Password", both containing placeholder text. At the bottom is a large, prominent blue button with the text "Sign in". The entire form is set against a light gray background.

Session Cookie

- Session Cookie is stored locally on User's Browser as a part of HTTP Response after successfully Authentication.
- Session Cookie is used to Authenticate User during subsequent HTTP Requests.
- This way User doesn't have to go through Login Form for each subsequent HTTP Request.
- Instead during each subsequent HTTP Request Session Cookie is automatically sent to the Web Application in order to Automatically Authenticate User (tell Server who made the HTTP Request).
- Session Cookie is assigned to a specific URL/Domain where Web Application that sent it lives
 - This means that when User sends another HTTP Request to <http://www.gmail.com/myemails> Session Cookie assigned to www.gmail.com is send with the HTTP Request to automatically Authenticate User into Gmail Web Application.
 - If User sends HTTP Request to some other URL/Domain like <http://www.yahoo.com/myemails> different Session Cookie, the one assigned to www.yahoo.com is send with the HTTP Request to automatically Authenticate User into Yahoo Web Application.

HTTP Response with a Session Cookie

```
POST /email/change HTTP/1.1
Host: vulnerable-website.com
Content-Type: application/x-www-form-urlencoded
Content-Length: 30
Cookie: session=yvthwsztyeQkAPzeQ5gHgTvlyxHfsAxE
email=wiener@normal-user.com
```

1.7.2 CSRF Attack - Theory

Info

- CSRF Attack allows **Attacker User** to trick the Web Application into thinking that he is the **Victim User**. And then issuing an action (like changing Password) without Victim's knowledge.
- But Attacker can't send HTTP Request to change Password to a Web Application directly from his Computer/Browser. This is because his Browser doesn't have a Session Cookie that will automatically Authenticate him as a Victim User. And without this Session Cookie Web Application wouldn't know who he is and will present him with a Login Form to Authenticate him. And since Attacker doesn't know Victim's Password it will not be able to Login as a Victim in order to change Victim's Password.
- So the idea is that the HTTP Request to Web Application to change the Password must come from Victim's Browser. Attacker must somehow trick the Victim to send HTTP Request to Web Application to change the Password. Attacker can do that by persuading Victim to visit **Attacker's Web Page** (Web Page made by the attacker). Attacker's Web Page will contain a Form that sends HTTP Request to a Gmail to change the Password.
- Instead of creating a fake Web Page with a Form that generates HTTP **POST** Request, sometimes Attacker can simple send a Link to the Victim as explained in **Attacker's Link** that sends HTTP **GET** Request. This can be done when Password can be changed through HTTP **GET** Request instead of HTTP **POST** Request.

Attacker.html

<https://www.gmail.com/changepassword?newpassword=mynewpassword>

```
<form method="GET" action="https://www.gmail.com/changepassword" >
  <input type="hidden" name="newpassword" value="mynewpassword" />
</form>
```

Attacker's Link

```

```

Attacker's Link

- Instead of creating a fake Web Page with a Form that generates HTTP Request, sometimes Attacker can simple send an URL to the Victim as shown below.
- This works when Password can be changed by sending HTTP GET Request since GET Request can be issued though a link. Unlike HTTP POST Request that can only be generated through the HTML Form.
- Links can be created
 - with `<a>` as text
 - with `` as image
- Attacker can simply place this link inside a Comment on some Post or as a part of his own Post.
When Victim clicks on a Text/Image Link HTTP GET Request is sent to Gmail together with Session Cookie to automatically Authenticate User and change his Password.

Attacker's Link

```

```

Attacker's Web Page

- Attacker knows that in order to change Gmail Password User must send HTTP Request in the following form

HTTP Request to change Password

```
https://www.gmail.com/changepassword?newpassword=mynewpassword
```

- User will never have to actually type this in.

Instead Gmail will give him a Form where User can enter new Password.

When User presses Submit this HTTP Request will be generated in the background and sent to the Gmail Web Application. Session Cookie will be automatically included into every HTTP Request that goes to www.gmail.com to Authenticate User.

- So now if Attacker wants to change User's Password he simply needs to create similar Form and persuade Victim to execute that Form from his Browser so that Session Cookie gets included into HTTP Request.

- So Attacker creates Attacker's Web Page with a Form shown below.

When a Victim visits Attacker's Web Page that Web Page gets loaded into Victim's Browser.

When victim presses Submit Button on the Form, that is part of this Web Page, Form will send HTTP Request to Gmail.

Since this HTTP Request is now being sent from Victim's Browser [Session Cookie](#) will automatically get included into the HTTP Request automatically Authenticating the User

Gmail will know that the action came from Authenticated User and will change the Password.

Although HTTP Request came from Authenticated User Gmail doesn't know that User did not want to perform this action.

Attacker.html

<https://www.gmail.com/changepassword?newpassword=mynewpassword>

```
<form method="GET" action="https://www.gmail.com/changepassword" >
  <input type="hidden" name="newpassword" value="mynewpassword" />
</form>
```

1.7.3 CSRF Attack - Application

Info

- This tutorial shows how to execute CSRF Attack against Spring Boot Application that doesn't have CSRF Token enabled.

1.7.4 CSRF Token - Theory

Info

- CSRF Token is one way of preventing CSRF Attack.
CSRF Token is a random value sent by Web App which is stored in a hidden Field inside the Form.
And which is then resent back to the Web App when User submits the Form.
This way Web App can make sure that HTTP Request came from the Web Page that was constructed by the Web App.
And not from the Web Page that was constructed by the Attacker.
We could say that **CSRF Token Authenticates Web Page**: From which Web Page is this HTTP Request coming?
- So every time Web Application sends a Web Page to a User, there will be a Form with a hidden Field whose value represents CSRF Token (and it will be different for each HTTP Response).
When User Submits the Form, this hidden Field is sent back to the Web App holding the value of CSRF Token.
Web App can now check if the value of received CSRF Token was sent previously to this User.
If so Web App knows that User used Web Page previously provided by the Web App to produce this HTTP Request.
- CSRF Token should be tied to a User.
That means that HTTP Request should be accepted only if it came with specific combination of User and CSRF Token.
User is defined in the Session Cookie.
CSRF Token is defined in the hidden Form Field.

CSRF Token - Vulnerabilities

- If CSRF Token is not properly implemented Web App can remain to be vulnerable to CSRF Attack.

Validation of CSRF Token depends on request method

- Some applications correctly validate the token when the request uses the POST method.
- But skip the validation when the GET method is used.
- In this situation, the attacker can switch to the GET method to bypass the validation and deliver a CSRF attack.

Validation of CSRF Token depends on token being present

- Some applications correctly validate the token when it is present but skip the validation if the token is omitted.
- In this situation attacker can simply remove the entire parameter containing the token (not just its value).

CSRF token is not tied to the user session

- Some applications do not validate that the token belongs to the same session/user who is making the request.
Instead application maintains a global pool of tokens that it has issued and accepts any token that appears in this pool.
- Attacker can log in to the application using their own account to obtain a valid token.
And then feed that token to the victim user in their CSRF attack.

1.7.5 CSRF Token - Application

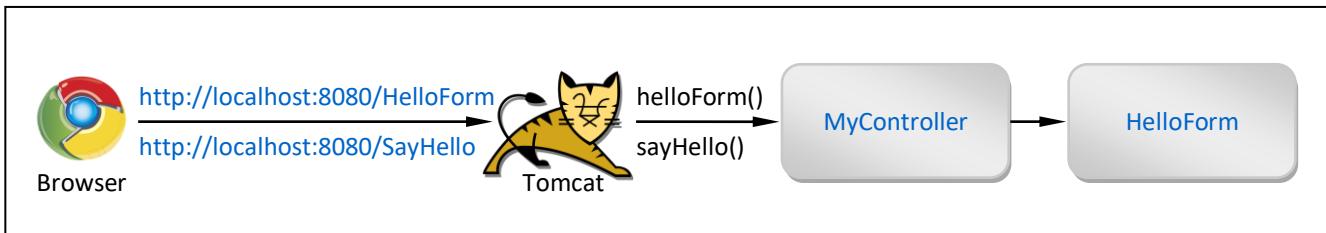
Info

[G]

- This tutorial shows how to enable CSRF Token for a Spring Boot Application.

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	Thymeleaf	Enables Controller to return reference to HTML Page HelloForm.html

Procedure

- **Create Project:** springboot_csrf_token_application (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)

application.properties

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

MyController.java

```
package com.ivoronline.springboot_csrf_token_application.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 {

    @RequestMapping("/HelloForm")
    public String helloForm() {
        return "HelloForm";
    }

    @ResponseBody
    @RequestMapping("/SayHello")
    public String sayHello(@RequestParam String name) {
        return "Hello " + name;
    }
}
```

HelloForm.html

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

    <title> HELLO FORM </title>

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

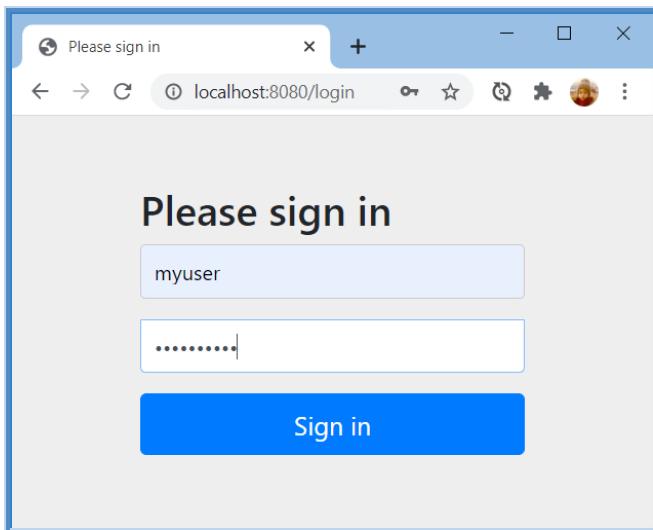
    <div>
        <h2> HELLO FORM </h2>
        <form method="GET" action="SayHello">
            <p> <input type="text" name="name" placeholder="Enter your name"/> </p>
            <p> <input type="submit" name="submit" style="width:100%" /> </p>
            <input type="hidden" th:name="${_csrf.parameterName}" th:value="${_csrf.token}"/>
        </form>
    </div>

</html>
```

Results

- <http://localhost:8080/HelloForm>
- You get redirected to <http://localhost:8080/login>
 - Username: myuser
 - Password: mypassword
 - Sign in
- You get redirected back to <http://localhost:8080/HelloForm>
 - (hidden `_csrf` Field gets populated by randomly generated CSRF Token `1e71147e-40e6-4653-956d-7346824089c3`)
 - Enter name: Bill
 - Submit
- You get redirected to <http://localhost:8080/SayHello> (with Form Input Parameters added to the URL)
 - http://localhost:8080/SayHello?name=Bill&submit=Submit&_csrf=1e71147e-40e6-4653-956d-7346824089c3
 - (hidden `_csrf` Field with CSRF Token is sent as part of HTTP Request `_csrf=1e71147e-40e6-4653-956d-7346824089c3`)

<http://localhost:8080/login>



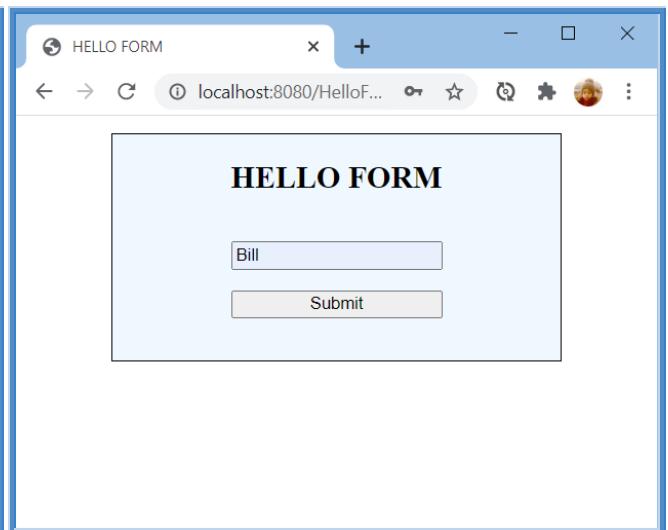
Please sign in

myuser

.....

Sign in

<http://localhost:8080>HelloForm>

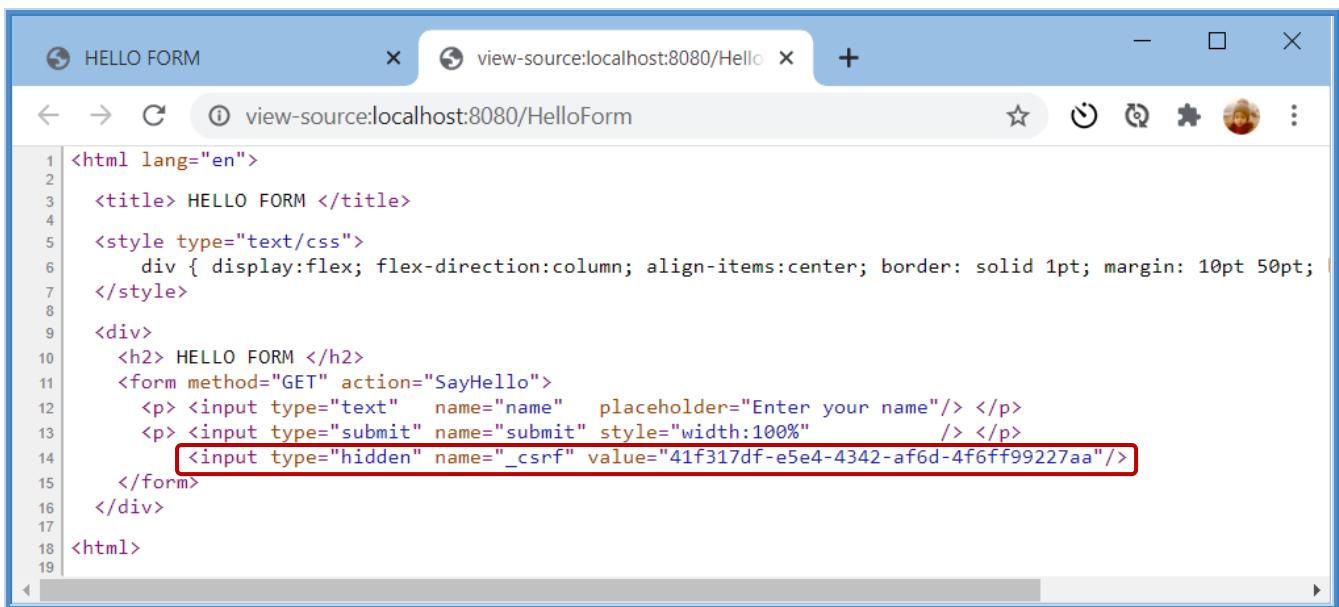


HELLO FORM

Bill

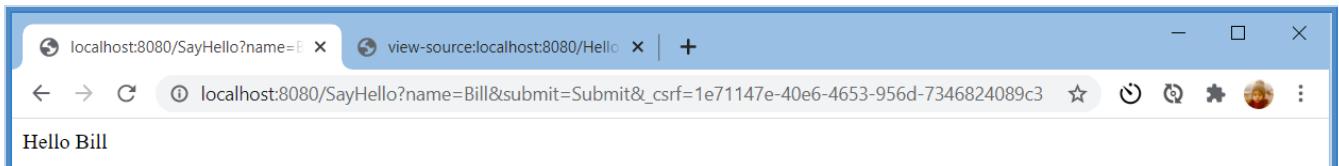
Submit

[View Source](#)



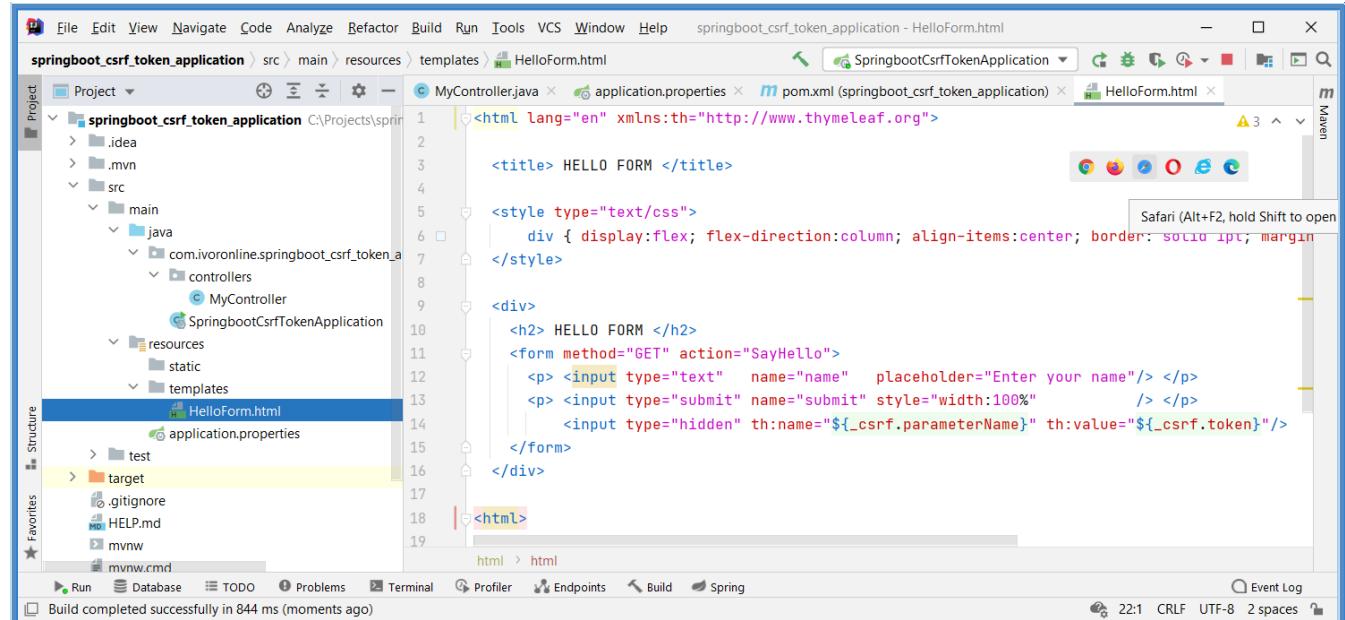
```
1 <html lang="en">
2
3   <title> HELLO FORM </title>
4
5   <style type="text/css">
6     div { display:flex; flex-direction:column; align-items:center; border: solid 1pt; margin: 10pt 50pt; }
7   </style>
8
9   <div>
10     <h2> HELLO FORM </h2>
11     <form method="GET" action="SayHello">
12       <p> <input type="text" name="name" placeholder="Enter your name"/> </p>
13       <p> <input type="submit" name="submit" style="width:100%" /> </p>
14       <input type="hidden" name="_csrf" value="41f317df-e5e4-4342-af6d-4f6ff99227aa"/>
15     </form>
16   </div>
17
18 </html>
```

http://localhost:8080/SayHello?name=Bill&submit=Submit&_csrf=1e71147e-40e6-4653-956d-7346824089c3



Hello Bill

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>
```

1.8 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.

1.8.1 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";
    }

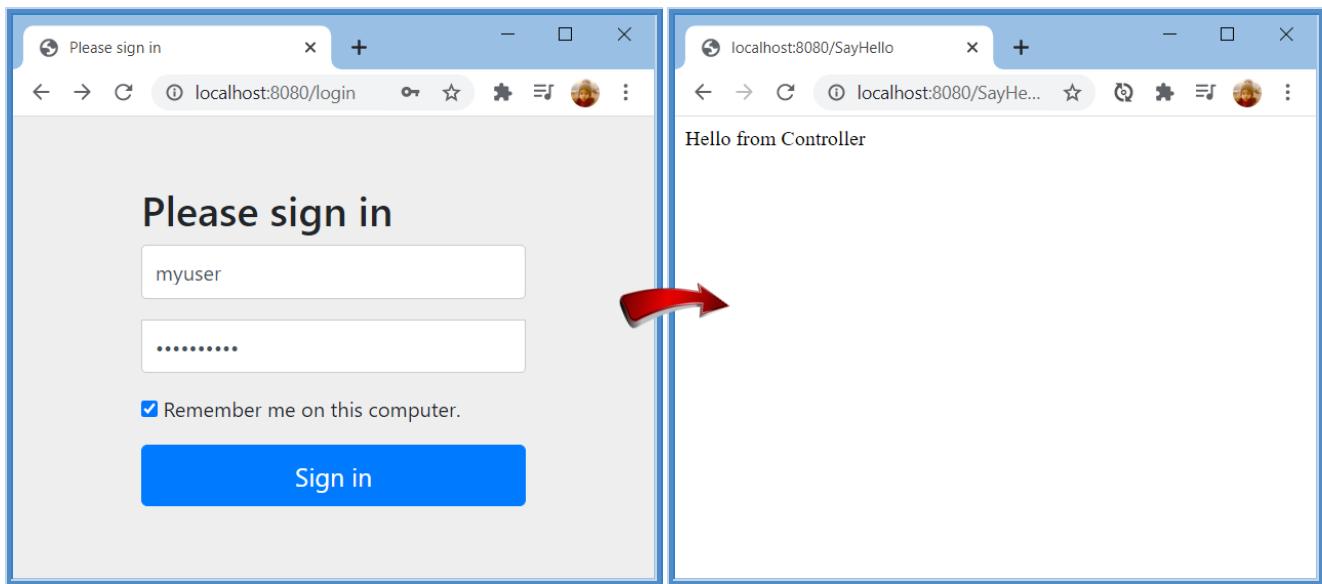
}
```

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

The screenshot shows the Google Chrome DevTools Application tab. The 'Cookies' section is highlighted with a red box. It lists a cookie named 'remember-me' with a value starting with 'bXl1c2VyOjE2MTI3...'. Other cookies listed include 'JSESSIONID' and 'http://localhost:8080'. The 'Application' tab is highlighted with a red box.

Name	Value	Domain	Expires / Max-Age	Size	Htt...	Sec...	Sam...	Prio...
JSESSIONID	DDD4CA96B5BC30...	localhost	Session	42	✓			Me...
remember-me	bXl1c2VyOjE2MTI3...	localhost	2021-02-08T10:18:36.76...	82	✓			Me...

Application Structure

```
java com ivoronline springboot_security_remmberme config SecurityConfig
```

```
@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();  
  
        //CUSTOM LOGIN FORM  
        httpSecurity.formLogin();  
    }  
}
```

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>
```

1.8.2 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

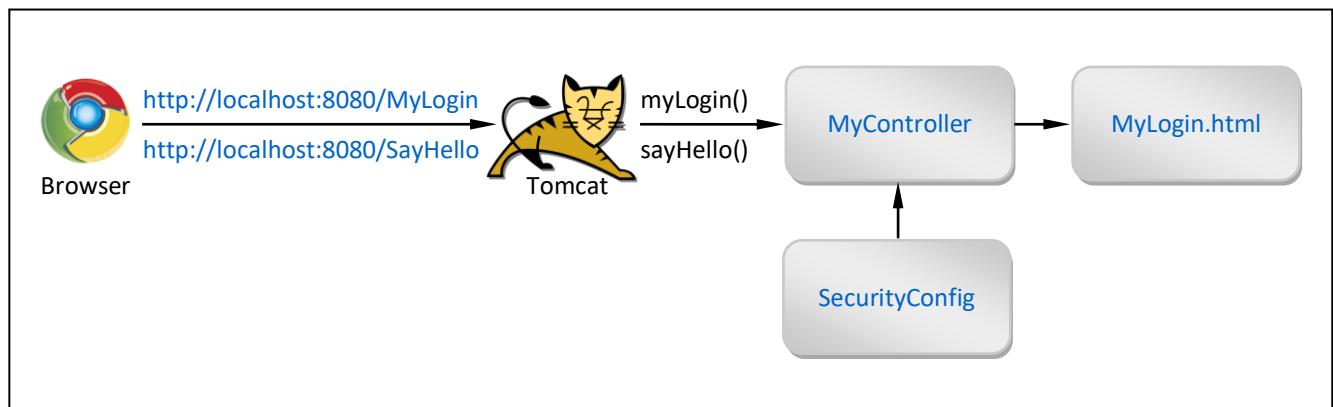
```
@RequiredArgsConstructor  
private final UserDetailsService userDetailsService;  
httpSecurity.rememberMe().key("something").userDetailsService(userDetailsService);
```

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	Thymeleaf	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>

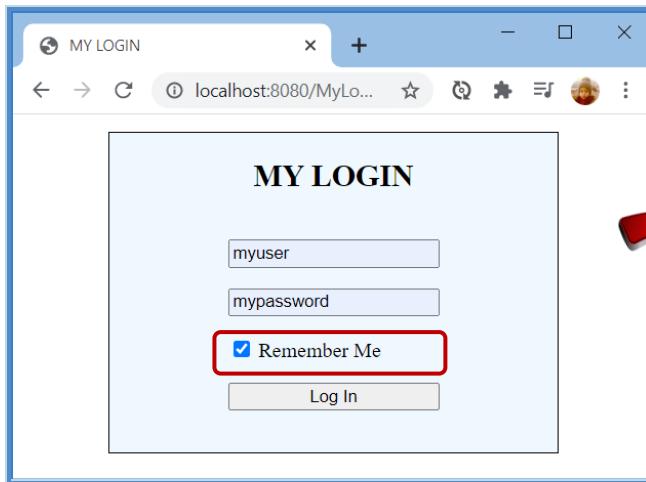
<style type="text/css">
    div { display:flex; flex-direction:column; align-items:center; border: solid 1pt; margin: 10pt 50pt;
background-color: aliceblue }
</style>

<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>
```

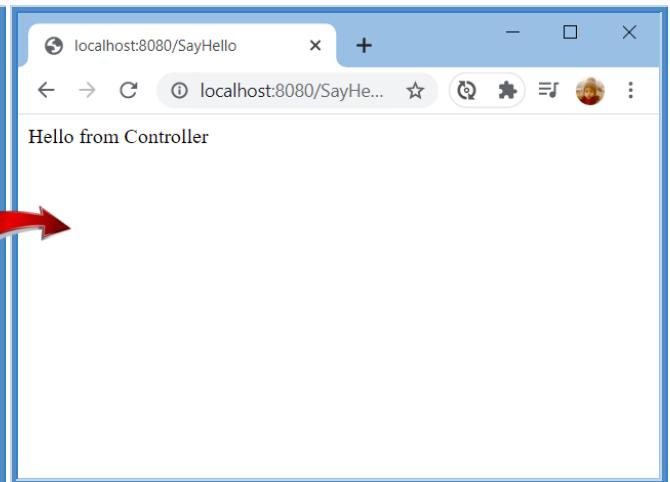
Results

- <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)

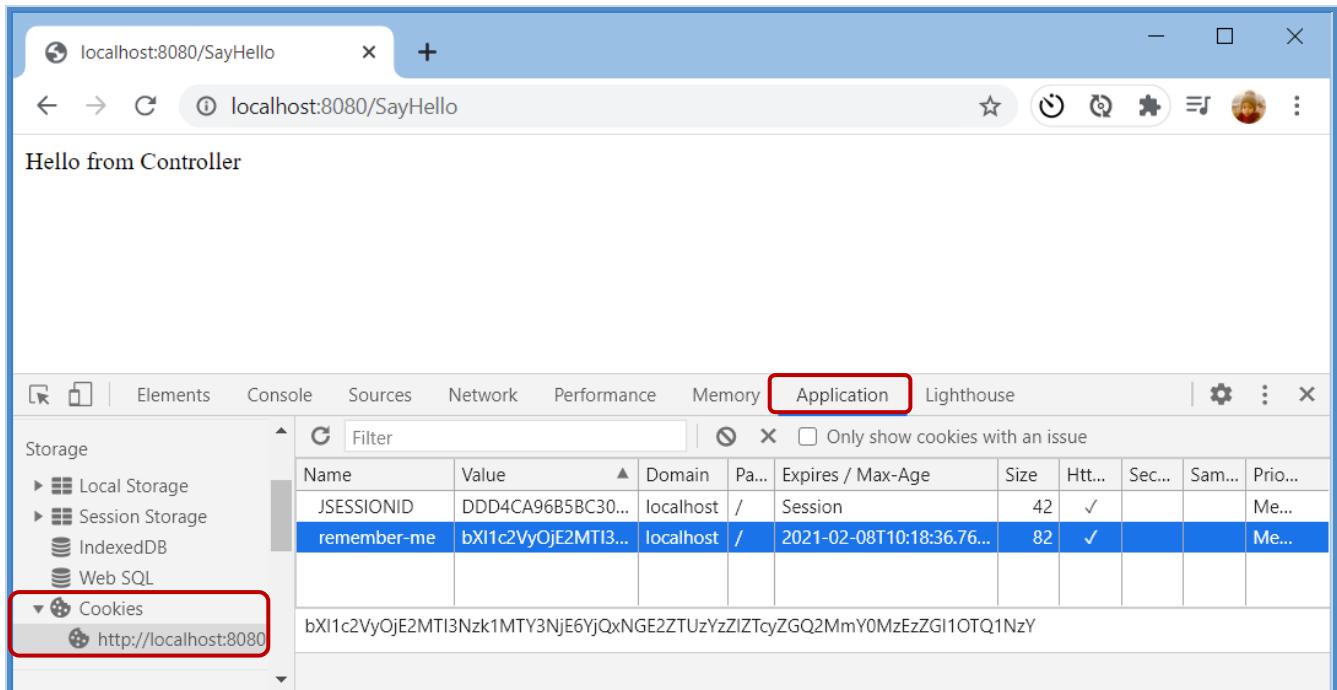
<http://localhost:8080/login>



<http://localhost:8080/SayHello>



remember-me Cookie



Name	Value	Domain	Path	Expires / Max-Age	Size	Http...	Sec...	SameSite	Prio...
JSESSIONID	DDD4CA96B5BC30...	localhost	/	Session	42	✓			Me...
remember-me	bXI1c2VyOjE2MTI3Nz... (highlighted)	localhost	/	2021-02-08T10:18:36.76...	82	✓			Me...

Application Structure

The screenshot shows the IntelliJ IDEA interface with the following details:

- Project Tree:** The project is named "springboot_security_rememberme_customform". It contains a .idea folder, .mvn folder, src folder with main and config subfolders, resources folder with static and templates subfolders containing MyLogin.html, and application.properties file.
- Code Editor:** The file "SecurityConfig.java" is open. The code defines a SecurityConfig class that extends WebSecurityConfigurerAdapter. It includes methods for configuring HttpSecurity, such as enabling remember-me cookies and disabling CSRF.
- Annotations:** A red box highlights the line of code: `httpSecurity.rememberMe().key("something").userDetailsService(userDetailsService);`.
- Status Bar:** Shows "All files are up-to-date (a minute ago)" and other build-related information like "41:1 CRLF UTF-8 2 spaces master".

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>
```

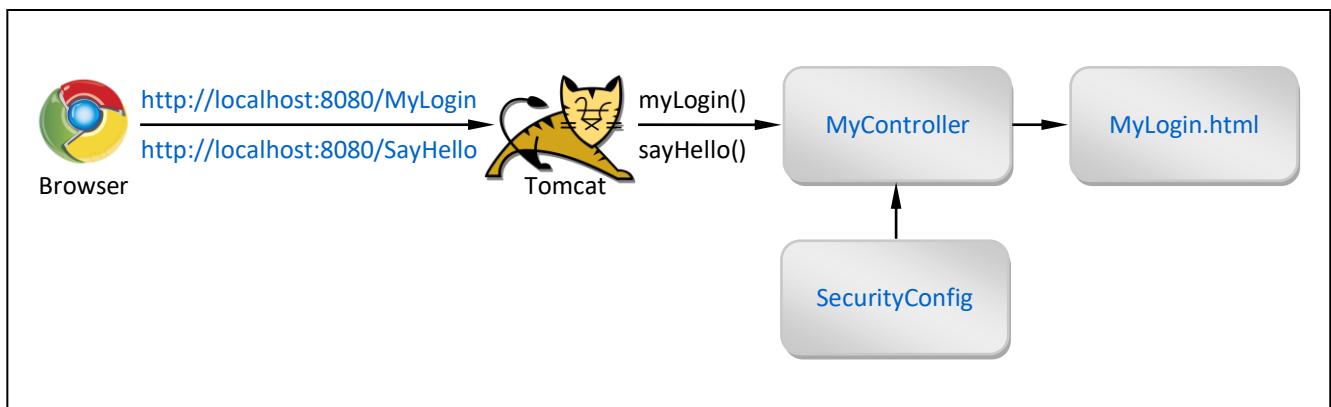
1.8.3 Login Form - Default - DB - PostgreSQL

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.ivorononline.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>

Please sign in

myuser

.....

Remember me on this computer.

Sign in

<http://localhost:8080/SayHello>

Hello from Controller

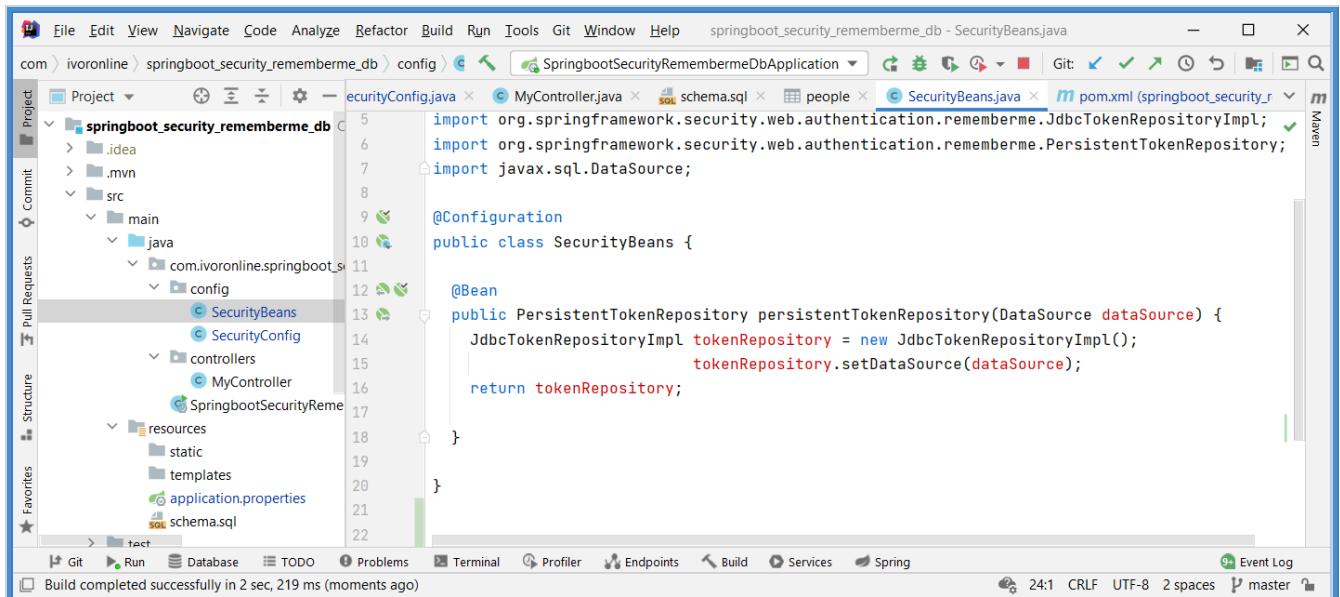
remember-me Cookie

Name	Value	Domain	Path	Expires / Max-Age	Size	Http...	Sec...	SameSite	Prio...
JSESSIONID	DDD4CA96B5BC30...	localhost	/	Session	42	✓			Me...
remember-me	bXl1c2VyOjE2MTI3Nzk1MTY3NjE6YjQxNGE2ZTUzYzZlZTcyZGQ2MmY0MzEzZGI1OTQ1NzY	localhost	/	2021-02-08T10:18:36.76...	82	✓			Me...

Persistent Token in DB Table: persistent_logins

persistent_logins				
Tx: Auto DB DDL DML Comma-delimited (CSV) Download Up Print				
Q <Filter Criteria>				
username	series	token	last_used	
1 myuser	esvmjf21+cB21PurJ31sTA==	B1zfMLI+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>
```

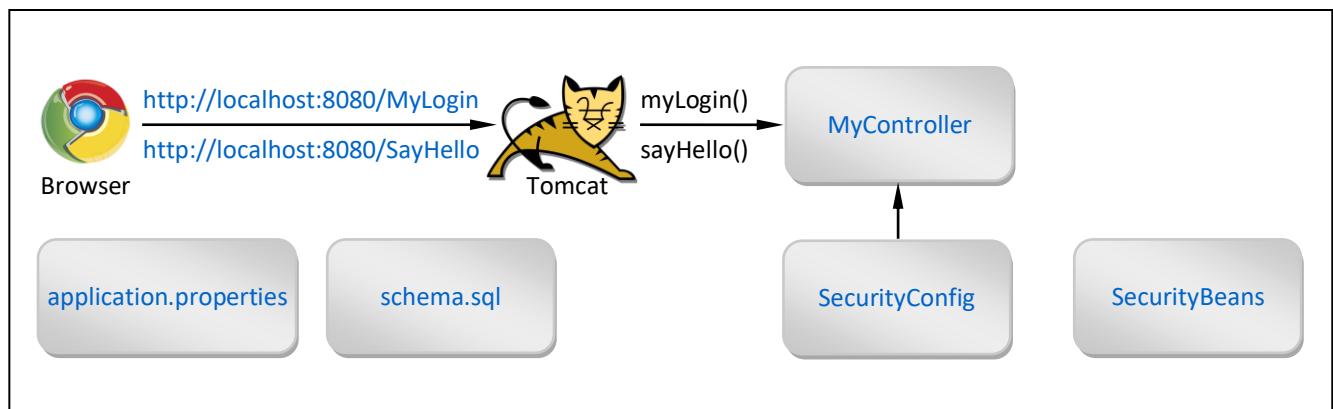
1.8.4 Login Form - Default - DB - H2

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>

Please sign in

myuser

.....

Remember me on this computer.

Sign in

<http://localhost:8080/SayHello>

Hello from Controller

remember-me Cookie

LocalStorage

SessionStorage

IndexedDB

Web SQL

Cookies

Name	Value	Domain	Path	Expires / Max-Age	Size	HttpOnly	Secure	SameSite	Priority
JSESSIONID	DDD4CA96B5BC30...	localhost	/	Session	42	✓			Me...
remember-me	bXl1c2VyOjE2MTI3...	localhost	/	2021-02-08T10:18:36.76...	82	✓			Me...

Persistent Token in DB Table: persistent_logins

The screenshot shows the H2 Console interface. On the left, the database structure is displayed with the PERSISTENT_LOGINS table expanded, showing columns: USERNAME, SERIES, TOKEN, and LAST_USED. On the right, a SQL query "SELECT * FROM PERSISTENT_LOGINS;" is run, and the results are shown in a table:

USERNAME	SERIES	TOKEN	LAST_USED
myuser	Bk9FpwLrV8oKwzvToPN+xQ==	44tTVwwMqasV+reZC3LcnQ==	2021-01-25 18:48:18.36

(1 row, 1 ms)

Application Structure

The screenshot shows the IntelliJ IDEA IDE. The left sidebar displays the project structure for "springboot_security_rememberme_db". The code editor shows the "SecurityBeans.java" file with the following content:

```
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;
    }
}
```

```
<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>
```

1.9 CORS (Cross Origin Resource Sharing)

Info

[R] [R]

- CORS stands for **Cross Origin Resource Sharing**.

CORS is when **Java Script** tries to access resource from another Origin/Application.

And not from Origin/Application which served the HTML Page with that Java Script.

- Origin/Application includes combination of **Protocol, Domain and Port**

- so if HTML containing Java Script was served from <http://localhost:8080>Hello.html>

- and Java Script from that Page tries to access image at <http://localhost:8085/SomeImage.png>

- then these two URLs represent different Origins/Applications (because they listen at different Ports 8080 vs 8085)

- Java Script runs inside Browsers which by default disables CORS.

So if your Spring Boot Application returns HTML Page that contains Java Script that needs to access resource from another Origin/Application then you need to **enable CORS on destination Application** (one that contains that resource).

- This means that in order to demonstrate CORS we need to create two Spring Boot Applications

- Source** Application will run on Port **8080** and it will contain HTML with **Java Script** calling Destination Application

- Destination** Application will run on Port **8085** and here we need to enable **CORS**

CORS Schema



1.9.1 Application - Source

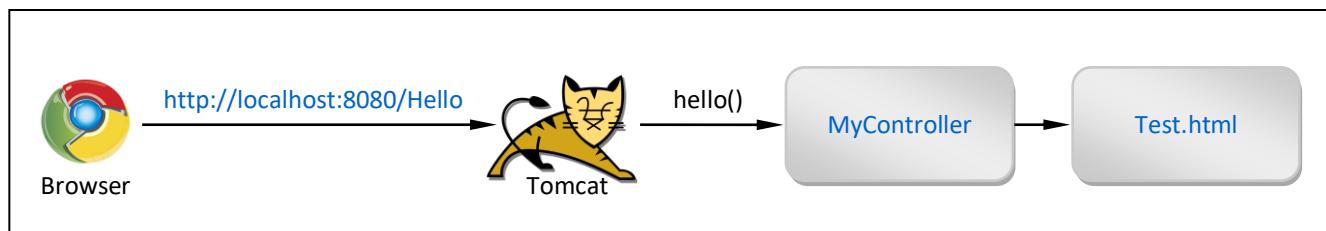
Info

[G]

- This tutorial shows how to create Source Application which will be used to demonstrate CORS functionality.
- Source Application** will run on Port **8080** and it will contain HTML with **Java Script** calling **Destination Application**.
- Source Application created in this tutorial will remain the same across subsequent tutorials.
This is because all CORS related changes only need to be done in Destination Application.
- It is Destination Application that needs to allow CORS access to its resources from other Origins/Applications.

Application Schema

[Results]



Spring Boot Starters

GROUP	DEPENDENCY	DESCRIPTION
Web	Spring Web	Enables @Controller, @RequestMapping and Tomcat Server.
Template Engines	Thymeleaf	Enables Controller to return reference to HTML Page <code>Test.html</code>

Procedure

- Create Project:** `springboot_cors_source` (add Spring Boot Starters from the table)
- Create Package:** `controllers` (inside main package)
 - Create Class:** `MyController.java` (inside package controllers)
- Create HTML File:** `Test.html` (inside directory resources/templates)

MyController.java

```
package com.ivoronline.springboot_cors_source.controllers;

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

@Controller
public class MyController {

    @GetMapping("/Hello")
    public String hello() {
        return "Test";
    }
}
```

Test.html

```
<title>Test.html</title>

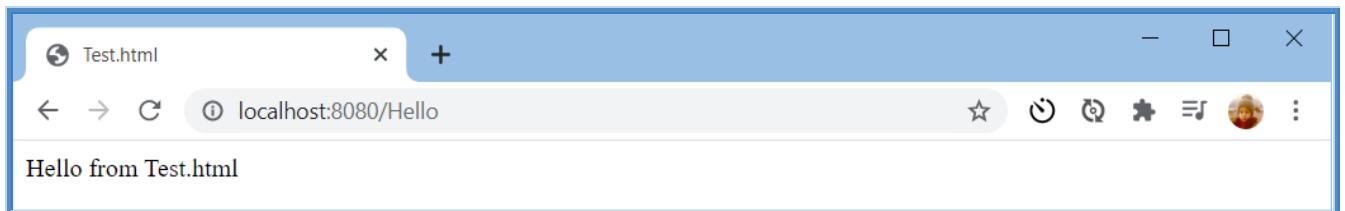
<p>Hello from Test.html</p>

<script>
    var request = new XMLHttpRequest();
    request.open('GET', 'http://www.ivoronline.com/Images/Banners/Banner_1.png');
    request.send();                                //This line throws CORS Error
</script>
```

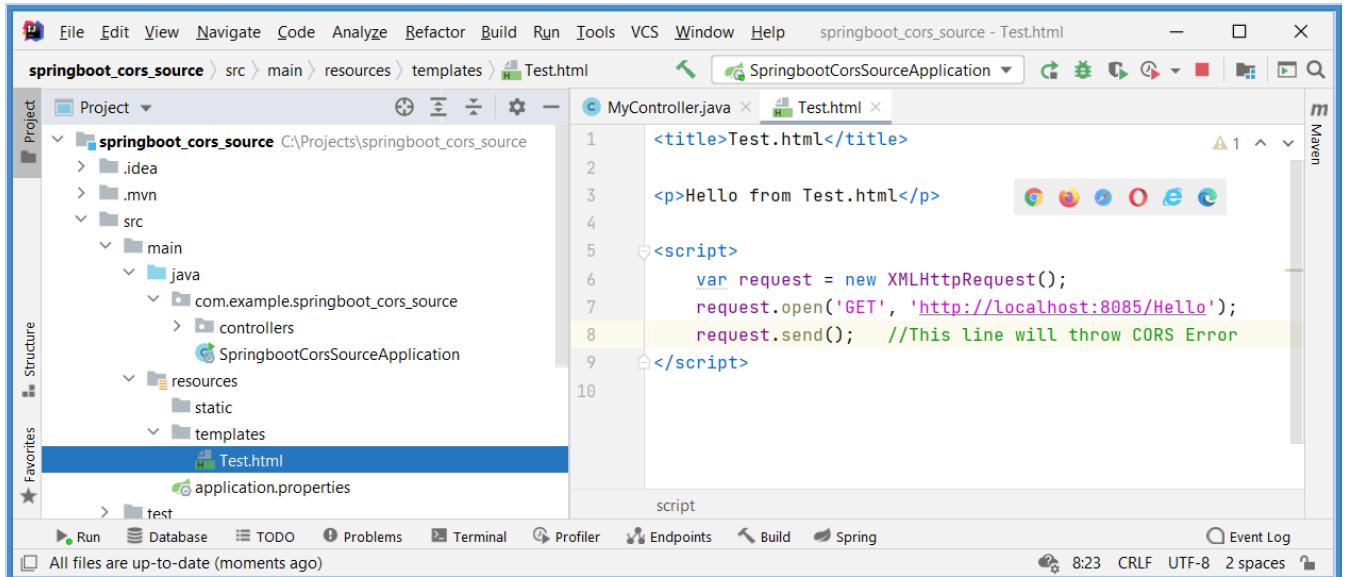
Results

<http://localhost:8080>Hello>

(just to confirm that Application is properly setup)



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-thymeleaf</artifactId>
    </dependency>

</dependencies>
```

1.9.2 Application - Destination

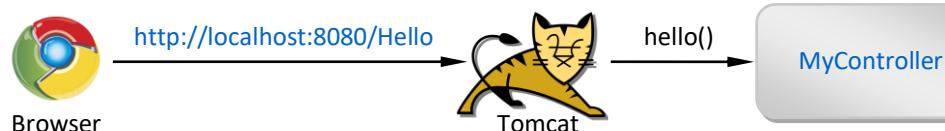
Info

[G]

- This tutorial shows how to create Destination Application which will be used to demonstrate CORS functionality.
- Destination Application** will run on Port **8085** and it will contain **Resource** that will be called by **Source Application**.
- Destination Application created in this tutorial will have CORS disabled (since this is default Browser's behavior). This means that this version of Application will be normal Spring Boot Application without any CORS functionality. Therefore initially Source Application will not be able to access Resource from Destination Application. In later tutorials we will show how to enable CORS in this Destination Application in different ways.

Application Schema

[Results]



Spring Boot Starters

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

Procedure

- **Create Project:** `springboot_cors_destination` (add Spring Boot Starters from the table)
- **Create Package:** `controllers` (inside main package)
 - **Create Class:** `MyController.java` (inside package controllers)

MyController.java

```
package com.ivoronline.springboot_cors_source.controllers;

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

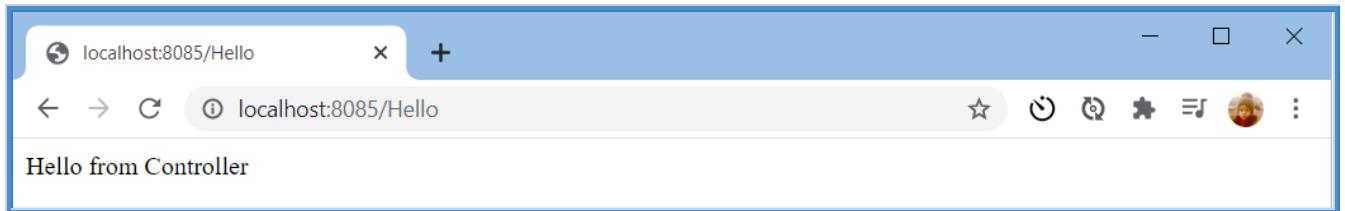
@Controller
public class MyController {

    @GetMapping("/Hello")
    public String hello() {
        return "Test";
    }
}
```

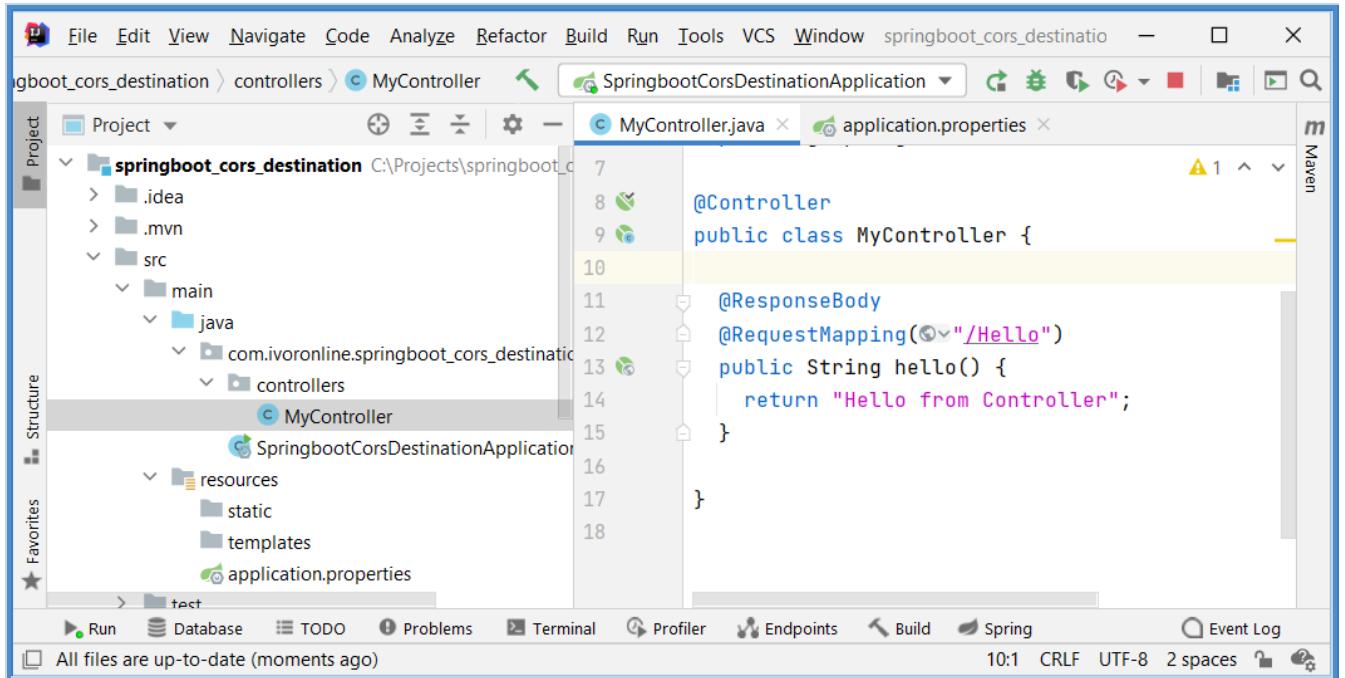
Results

<http://localhost:8085>Hello>

(just to confirm that Application is properly setup)



Application Structure



pom.xml

```
<dependencies>

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

</dependencies>
```

1.9.3 CORS - Disabled

Info

- This tutorial shows how
 - [Source Application throws CORS Error](#) (we will use Google Chrome Developer Tools to see CORS Error)
 - when trying to access Resource from Destination Application
 - when **CORS is by default disabled** in [Destination Application](#)
- Since this is default behavior of Spring Boot Application no changes will be made to [Destination Application](#).

CORS Schema

[[Results](#)]



Results

- Start Source Application
- Start Destination Application
- <http://localhost:8080>
- Customize and control Google Chrome
 - More Tools
 - Developer Tools
 - Network

<http://localhost:8080>Hello>

(throws CORS error)

The screenshot shows the Google Chrome developer tools Network tab. A red arrow points from the top right of the browser window to the three-dot menu icon in the toolbar. Another red arrow points from the bottom right of the Network tab to the 'Initiator' column of the request table.

Name	Status	Type	Initiator	Size	Time	Waterfall
Hello	200	document	Other	440 B	7 ms	
Hello	CORS error	xhr	Hello:8	0 B	117 ms	

2 requests | 440 B transferred | 235 B resources | Finish: 142 ms | DO contentLoaded: 31 ms | Load: 39 ms

Click on Initiator Hello:8

The screenshot shows the Google Chrome developer tools Sources tab. A red arrow points from the bottom right of the Network tab down to the 'Hello' file in the Sources tree. The code editor shows a script block with a comment indicating where the error occurs:

```
1 <title>Test.html</title>x
2
3 <p>Hello from Test.html</p>
4
5 <script>
6   var request = new XMLHttpRequest();
7   request.open('GET', 'http://localhost:8085>Hello');
8   request.send(); //This Line throws CORS Errorx
9 </script>
10
```

Line 8, Column 13

1.9.4 CORS - Enabled - Annotations

Info

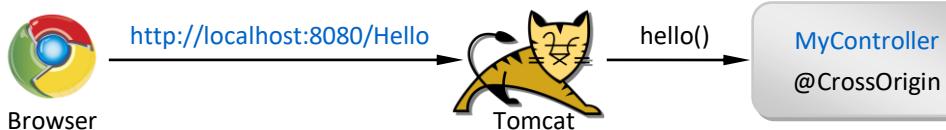
- This tutorial shows how to enable CORS in Destination Application by using Annotations.

MyController.java

```
@CrossOrigin
```

Destination Application Schema

[Results]



Changes in Destination Application

- Edit Class: MyController.java

MyController.java

```
package com.ivoronline.springboot_cors_destination.controllers;

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

@Controller
public class MyController {

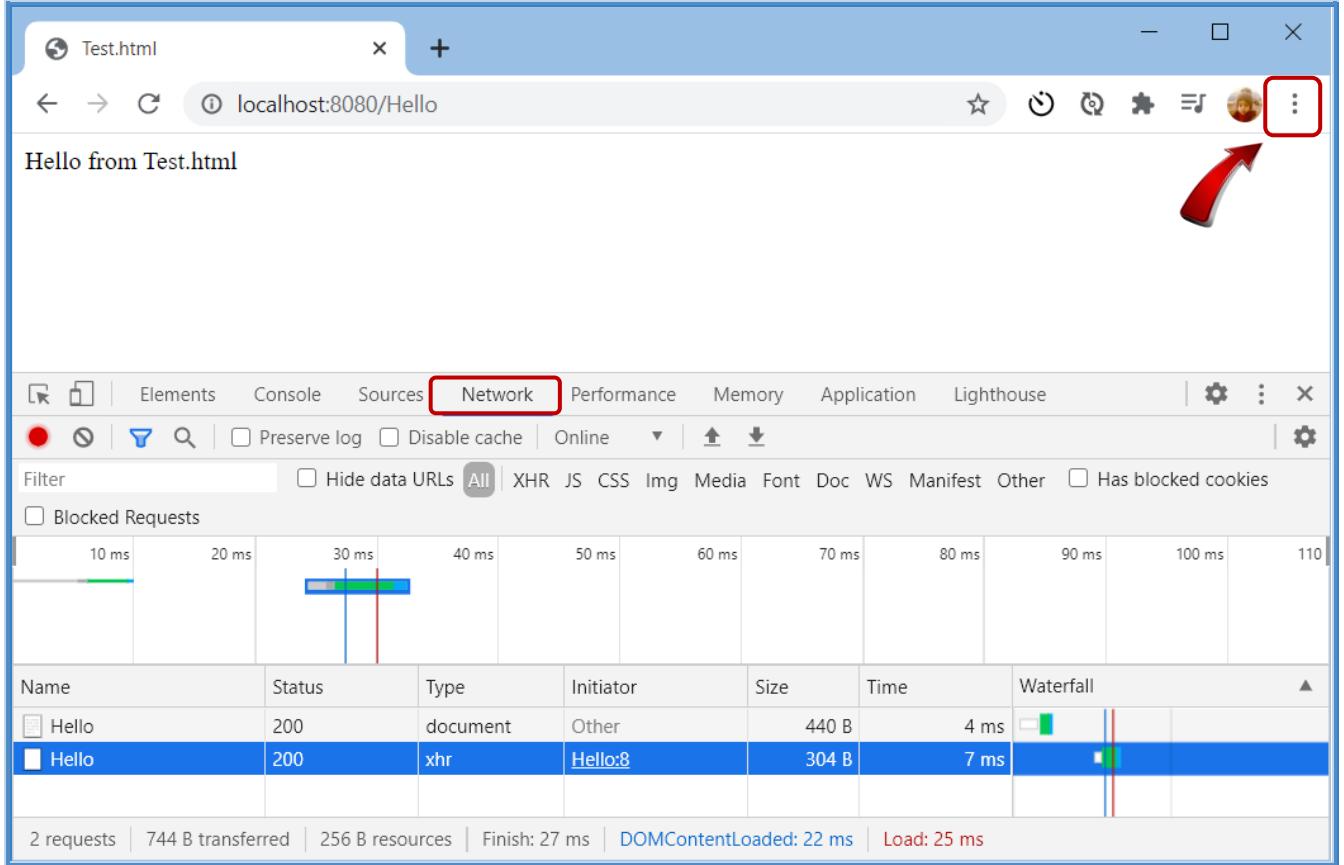
    @CrossOrigin
    @ResponseBody
    @RequestMapping("/Hello")
    public String hello() {
        return "Hello from Controller";
    }
}
```

Results

- Start Source Application
- Start Destination Application
- <http://localhost:8080>
- Customize and control Google Chrome
 - More Tools
 - Developer Tools
 - Network

<http://localhost:8080>Hello>

(no CORS error)



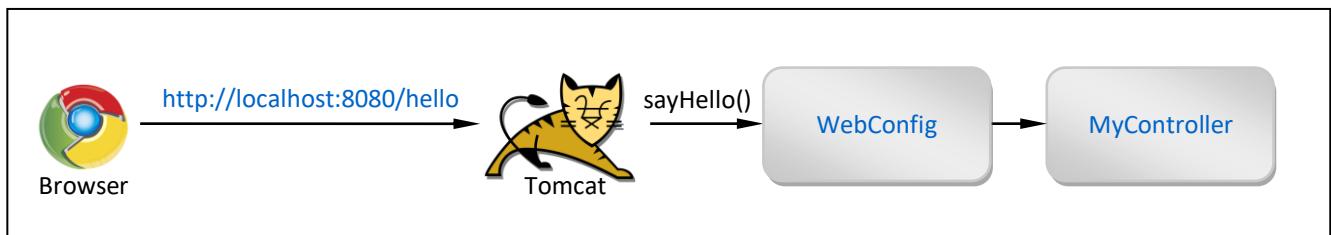
1.9.5 CORS - Enabled - API

Info

- This tutorial shows how to enable CORS in Destination Application by using API in newly created Class WebConfig.java.

Destination Application Schema

[Results]



Changes in Destination Application

- Create Package: config (inside main package)
 - Create Class: WebConfig.java (inside package config)

WebConfig.java

```
package com.ivoronline.springboot_cors_destination.config;

import org.springframework.context.annotation.Configuration;
import org.springframework.web.servlet.config.annotation.CorsRegistry;
import org.springframework.web.servlet.config.annotation.WebMvcConfigurer;

@Configuration
public class WebConfig implements WebMvcConfigurer {

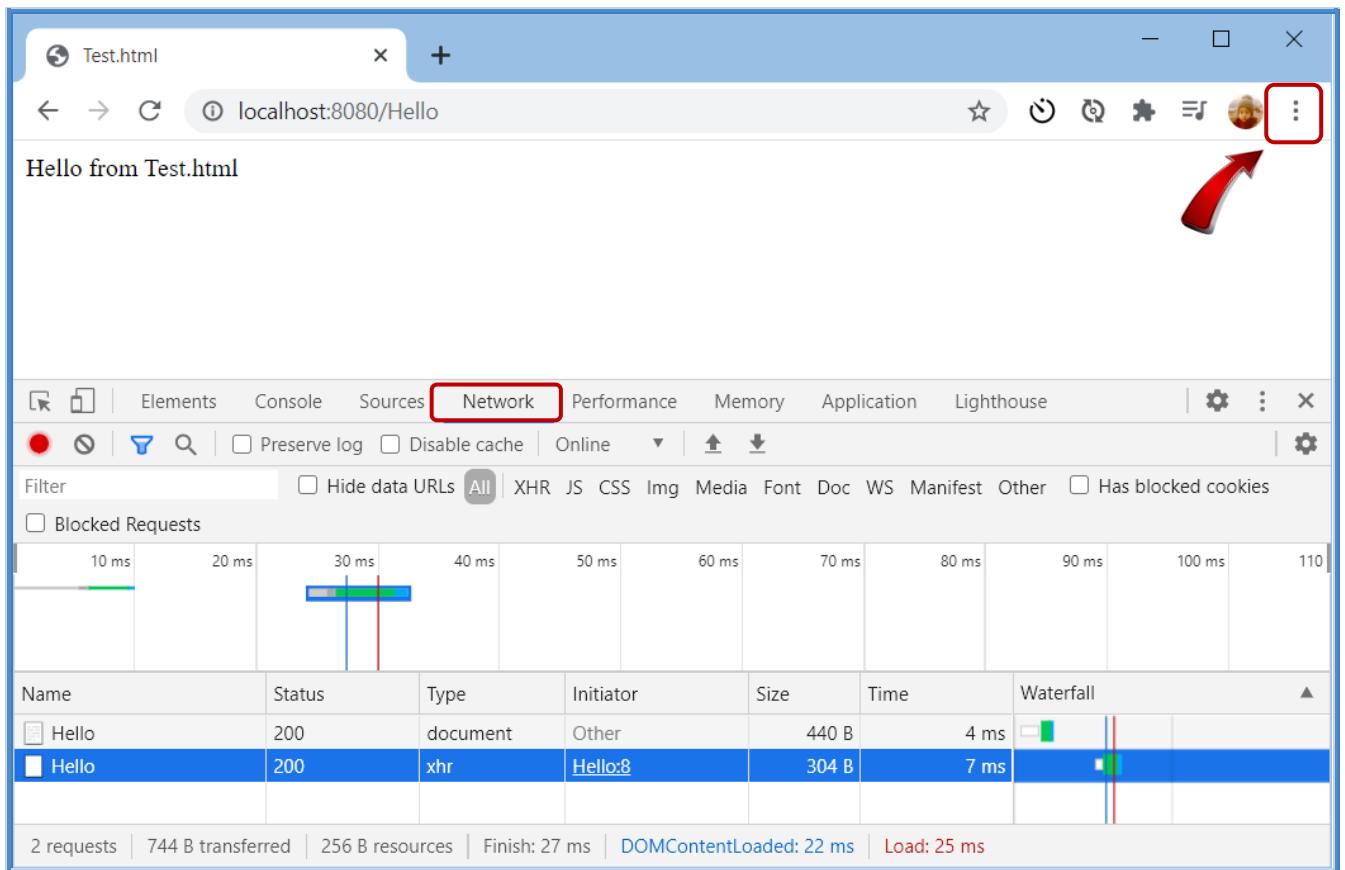
    @Override
    public void addCorsMappings(CorsRegistry registry) {
        registry.addMapping("/Hello").allowedMethods("GET", "POST", "PUT");
    }
}
```

Results

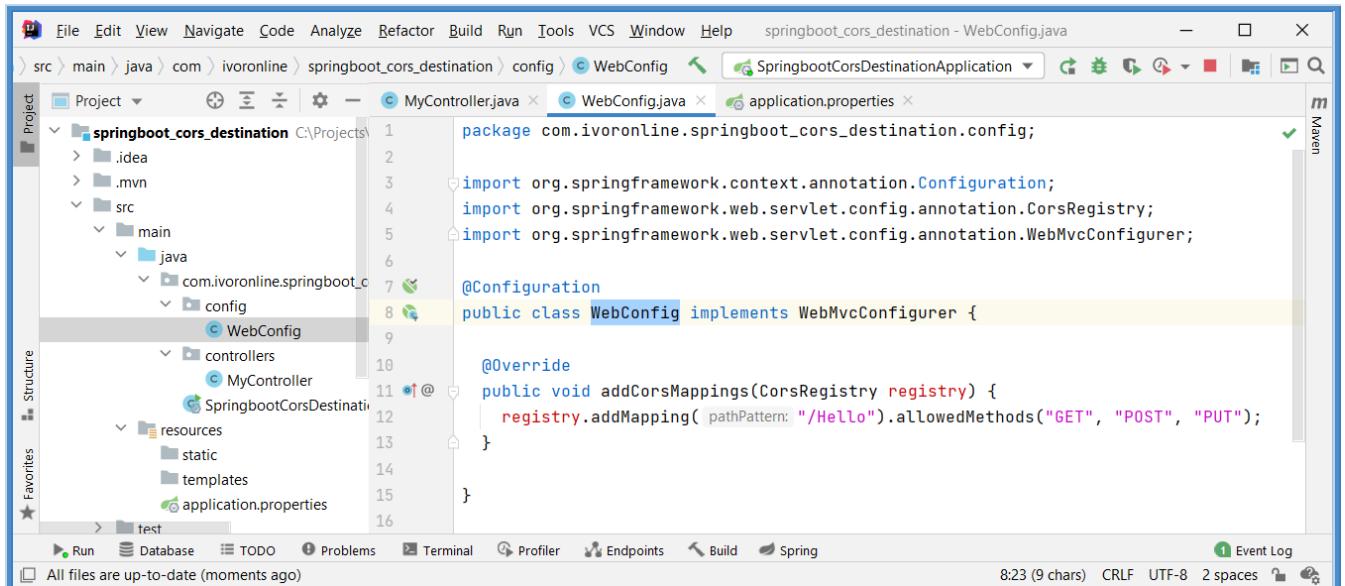
- Start Source Application
- Start Destination Application
- <http://localhost:8080>
- Customize and control Google Chrome
 - More Tools
 - Developer Tools
 - Network

<http://localhost:8080>Hello>

(no CORS error)



Destination Application Structure



1.10 2FA (2 Factor Authentication)

Info

[G]

- Following tutorials show how to implement 2FA (2 Factor Authentication).
- 2 Factor Authentication is called that way because User must Authenticate twice
 - first he must provide his **Username** and **Password**
 - then he must enter a **Code** sent to him by Google Authentication Service
- To enable 2FA we will add following Properties to Account Entity
 - Boolean `google2faEnabled = true` indicates that Account now has Google Secret Key and it can start using 2FA Service. This is triggered by calling Endpoint <http://localhost:8080/CodeEnter> (and before it returns HTML Page with QRCode). User can now scan QRCode with specialized Mobile App to get his Account into the Mobile App. After that User can simply open Mobile App to get Temporary Code any time he wants to Login again.

Process Flow

- User tries to access restricted URL <http://localhost:8080>Hello>
- User is redirected to Login Page <http://localhost:8080/login>
 - User enters his Username and Password
- User is redirected to QR Code Page <http://localhost:8080/QRCode>
 - User scans QR Code with his Mobile Phone
 - User is redirected to Google Authentication Page which shows him a Code (this Code is valid only for him for 30s)
 - User enter Code into QR Code Page
 - User presses Submit Button
- User is redirected to Endpoint that verifies entered Code <http://localhost:8080/VerifyCode>

Application Structure

- Following tutorials are based on [User - From DB](#) tutorial as starting point (which uses Accounts stored in DB). Compared to [User - From DB](#) tutorial we will need additional Spring Boot Starters highlighted below. Their respective Maven dependencies are shown in below `pom.xml`.

`pom.xml`

```
<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>
```

1.10.1 Step 1: Copy Project

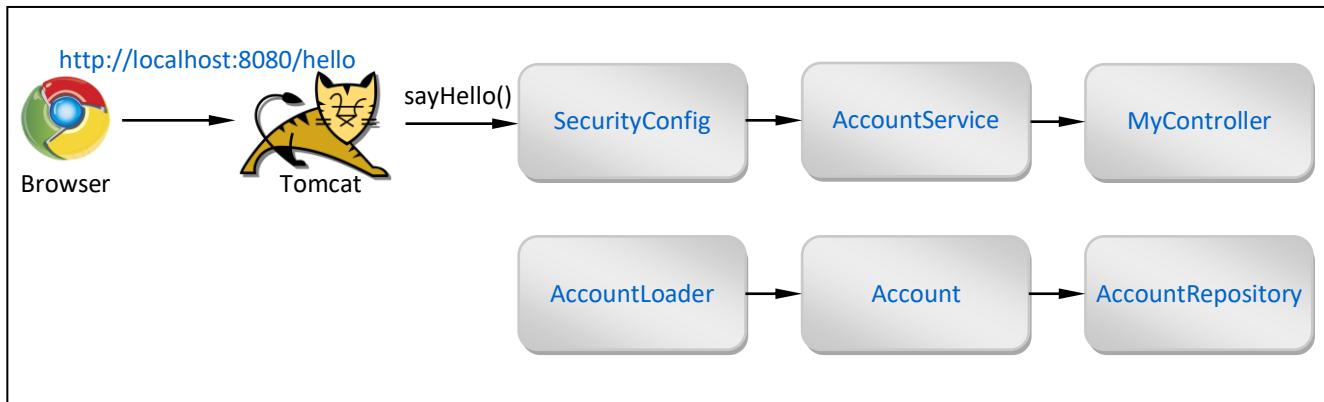
Info

[G]

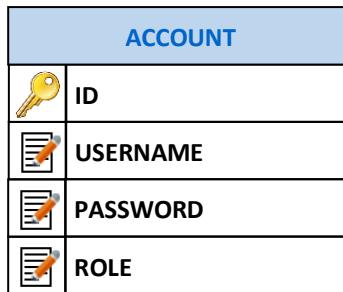
- Since we want to use [User - From DB tutorial](#) as starting point, this tutorial shows how to create new Project, copy content of [User - From DB tutorial](#) into it and then test functionality to make sure that everything works correctly.

Application Schema

[Results]



DB Schema



DB Table: Account

(Loaded Data)

ID	USERNAME	PASSWORD	ROLE
1	john	johnpassword	ROLE_ADMIN

Spring Boot Starters

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

Procedure

- Create Project: `springboot_security_2f` (add Spring Boot Starters from the table)
- Copy Project: `User - From DB` (overwrite src Directory, rename main Package & Class to reflect new Name)
- Edit File: `application.properties` (specify PostgreSQL DB)

application.properties

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

# JPA / HIBERNATE
spring.jpa.hibernate.ddl-auto = create-drop
```

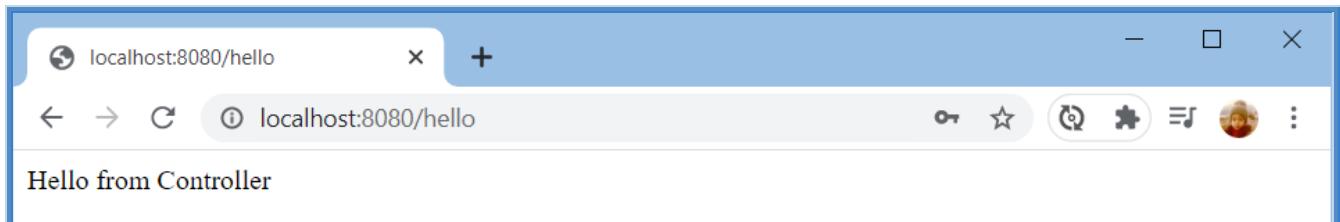
Results

- <http://localhost:8080/hello>
- You get redirected to <http://localhost:8080/login>
 - Username: **john**
 - Password: **johnpassword**
 - Sign in
- You get redirected back to <http://localhost:8080/hello>

<http://localhost:8080/login> - john - johnpassword



<http://localhost:8080/hello>



DB Table: Account

(Loaded Data)

	id	password	role	username
1	1	1 johnpassword	ROLE_ADMIN	john

Application Structure

The screenshot shows the IntelliJ IDEA interface. The left sidebar displays the project structure under 'Project'. The 'src' folder contains 'main' and 'test' packages. 'main' has 'java', 'config', 'controllers', 'entities', 'repositories', and 'services' sub-folders. 'services' contains 'AccountService'. The code editor on the right shows the implementation of 'AccountService.java'. The code uses Lombok annotations like @Service and @Autowired. It interacts with 'AccountRepository' to load a user by username and creates authorities and users. The bottom status bar shows the file path as 'C:\Projects\springboot_security_2fa\AccountService.java', the file name as 'AccountService.java', and other details like '29:55 CRLF UTF-8 2 spaces'.

```
16  @Service
17  public class AccountService implements UserDetailsService {
18
19      @Autowired
20      AccountRepository accountRepository;
21
22      @Override
23      public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {
24
25          //GET ACCOUNT FROM DB
26          Account account = accountRepository.findByUsername(username);
27
28          //CREATE AUTHORITIES
29          List<GrantedAuthority> authorities = new ArrayList<GrantedAuthority>();
30          authorities.add(new SimpleGrantedAuthority(account.getRole()));
31
32          //CREATE USER
33          User user = new User(account.getUsername(), account.getPassword(), enabled: true, account.getAuthorities());
34
35          //RETURN USER
36          return user;
37
38      }
39  }
```

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>

    <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>
```

1.10.2 Step 2: Register

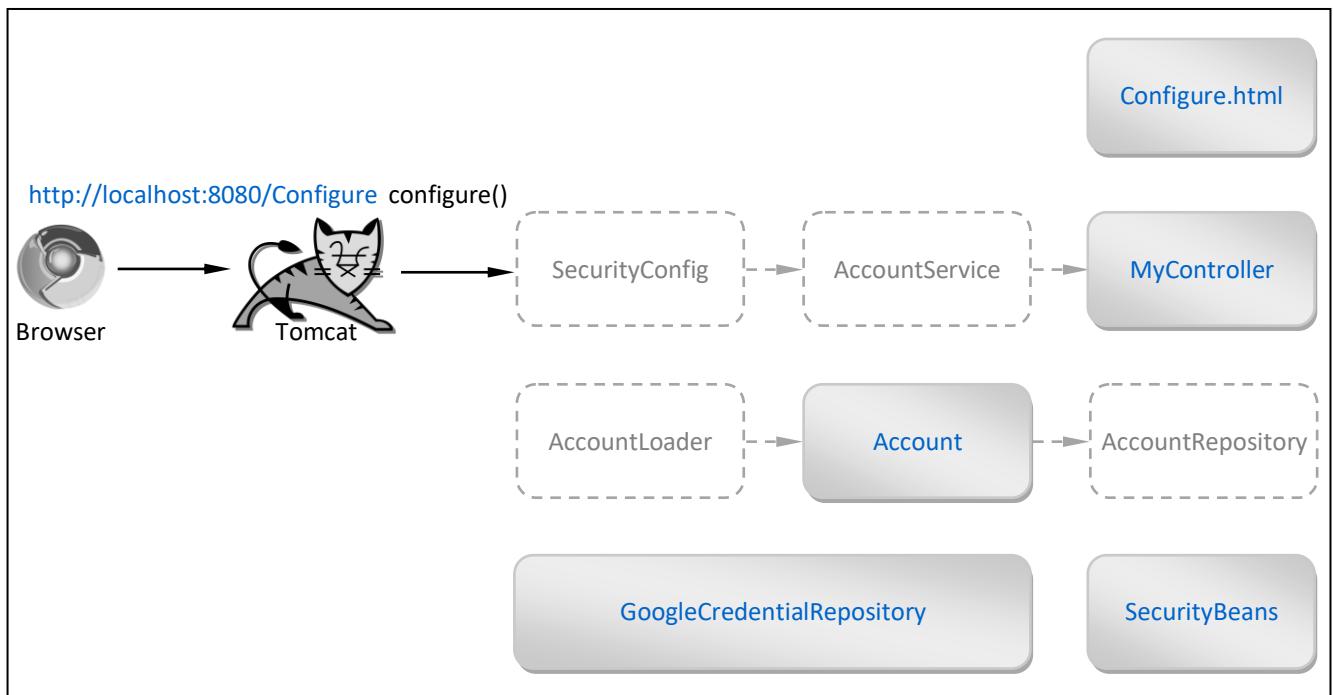
Info

[G]

- This tutorial shows how to create Endpoint that
 - registers User for Google 2FA Service
 - displays HTML Page that shows **Google QR Code** (needed to initialize Mobile App to display Temporary Code)

Application Schema

[Results]



Show QR Code

- Edit File: [pom.xml](#) (add Maven Dependency for googleauth)
- Edit Class: [Account.java](#) (add Properties related to Google 2FA)
- Create Class: [GoogleCredentialRepository.java](#) (inside package config)
- Create Class: [SecurityBeans.java](#) (inside package config)
- Edit Class: [MyController.java](#) (inside package controllers)
- Create File: [Configure.html](#) (inside directory resources/templates)

pom.xml

```
<dependency>
    <groupId>com.warrenstrange</groupId>
    <artifactId>googleauth</artifactId>
    <version>1.5.0</version>
</dependency>
```

Account.java

```
package com.ivoronline.springboot_security_2fa.entities;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;

@Entity
public class Account {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    public Integer id;
    public String username;
    public String password;
    public String role;

    public String google2faSecretKey;
    public Boolean google2faEnabled = false;
    public Boolean google2faAuthenticated = false;

}
```

GoogleCredentialRepository.java

```
package com.ivorononline.springboot_security_2fa.config;

import com.ivorononline.springboot_security_2fa.entities.Account;
import com.ivorononline.springboot_security_2fa.repositories.AccountRepository;
import com.warrenstrange.googleauth.ICredentialRepository;
import lombok.RequiredArgsConstructor;
import org.springframework.stereotype.Component;
import java.util.List;

@RequiredArgsConstructor
@Component
public class GoogleCredentialRepository implements ICredentialRepository {

    private final AccountRepository accountRepository;

    //=====
    // SAVE USER CREDENTIALS (STORE SECRET KEY)
    //=====

    @Override
    public void saveUserCredentials(String userName, String secretKey, int validationCode, List<Integer> scratchCodes) {

        //GET ACCOUNT
        Account account = accountRepository.findByUsername(userName);
        account.google2faSecretKey = secretKey;
        account.google2faEnabled = true;

        //STORE ACCOUNT
        accountRepository.save(account);

    }

    //=====
    // GET SECRET KEY
    //=====

    @Override
    public String getSecretKey(String userName) {

        //GET ACCOUNT
        Account account = accountRepository.findByUsername(userName);

        //RETURN SECRET KEY
        return account.google2faSecretKey;

    }

}
```

SecurityBeans.java

```
package com.ivorononline.springboot_security_2fa.config;

import com.warrenstrange.googleauth.GoogleAuthenticator;
import com.warrenstrange.googleauth.GoogleAuthenticatorConfig;
import com.warrenstrange.googleauth.ICredentialRepository;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import java.util.concurrent.TimeUnit;

@Configuration
public class SecurityBeans {

    @Bean
    public GoogleAuthenticator googleAuthenticator(ICredentialRepository credentialRepository){

        GoogleAuthenticatorConfig.GoogleAuthenticatorConfigBuilder configBuilder = new
        GoogleAuthenticatorConfig.GoogleAuthenticatorConfigBuilder();

        configBuilder
            .setTimeStepSizeInMillis(TimeUnit.SECONDS.toMillis(30))
            .setWindowSize(10)
            .setNumberOfScratchCodes(0);

        GoogleAuthenticator googleAuthenticator = new GoogleAuthenticator(configBuilder.build());
        googleAuthenticator.setCredentialRepository(credentialRepository);

        return googleAuthenticator;
    }
}
```

MyController.java

```
package com.ivorononline.springboot_security_2fa.controllers;

import com.ivorononline.springboot_security_2fa.repositories.AccountRepository;
import com.warrenstrange.googleauth.GoogleAuthenticator;
import com.warrenstrange.googleauth.GoogleAuthenticatorKey;
import com.warrenstrange.googleauth.GoogleAuthenticatorQRGenerator;
import lombok.RequiredArgsConstructor;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.security.core.userdetails.User;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.ResponseBody;

@Controller
@RequiredArgsConstructor
public class MyController {

    private final AccountRepository accountRepository;
    private final GoogleAuthenticator googleAuthenticator;

    //=====
    // CONFIGURE (SERVICE & MOBILE APP)
    //=====

    @RequestMapping("/Configure")
    public String configure(Model model){

        //GET USERNAME
        User user = (User) SecurityContextHolder.getContext().getAuthentication().getPrincipal();
        String username = user.getUsername();

        //CREATE NEW KEY IN DB (everytime it is called)
        GoogleAuthenticatorKey key = googleAuthenticator.createCredentials(username);

        //CONSTRUCT URL FOR QR CODE IMAGE
        String googleURL = GoogleAuthenticatorQRGenerator.getOtpAuthURL("something", username, key);

        //ADD GOOGLE URL AS INPUT PARAMETER TO HTML PAGE
        model.addAttribute("googleURL", googleURL);

        //RETURN CONFIRMATION HTML PAGE
        return "Configure";
    }

    //=====
    // Hello
    //=====

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

Configure.html

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

<p>  </p>
```

Results

- <http://localhost:8080/Configure>
- You get redirected to <http://localhost:8080/login>
 - Username: **john**
 - Password: **johnpassword**
 - Sign in
- You get redirected back to <http://localhost:8080/Configure> (shows QRCode)
 - Scan QRCode with Google Authenticator (Mobile Phone App)
 - It should show a Temporary Code: **267 189**

<http://localhost:8080/login> - john - johnpassword



<http://localhost:8080/Configure>

Scanning with **Google Authenticator** Mobile App

Two screenshots side-by-side. The left screenshot shows a QR code on a web page titled "localhost:8080/Configure". The right screenshot shows the Google Authenticator app interface with the text "something (john)" and the temporary code "267 189".

Application Structure

The screenshot shows the IntelliJ IDEA interface with the following details:

- Project View:** Shows the project structure under "springboot_security_2fa". The "src" folder contains "main" and ".mvn" subfolders. "main" has "java", "config", and "controllers" subfolders. "java" contains "com.ivoronline.springboot_security_2fa" package with "AccountLoader", "GoogleCredentialRepository", "SecurityBeans", "SecurityConfig", and "MyController" classes. "config" contains "AccountRepository" and "AccountService". "resources" contains "static" and "templates" folders with "Configure.html" and "application.properties".
- Code Editor:** The "MyController.java" file is open. It contains Java code for a controller method named "configure". The code uses annotations like @RequestMapping and @ModelAttribute. It interacts with the SecurityContextHolder to get the authenticated user's principal and creates a GoogleAuthenticatorKey. It constructs a QR code URL and adds it to a Thymeleaf model. Finally, it returns a confirmation HTML page.
- Toolbars and Status Bar:** The status bar at the bottom shows "All files are up-to-date (5 minutes ago)" and the current time as 17:14.

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>

    <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>

    <dependency>
        <groupId>com.warrenstrange</groupId>
        <artifactId>googleauth</artifactId>
        <version>1.5.0</version>
    </dependency>
</dependencies>
```

1.10.3 Step 3: Enter Code

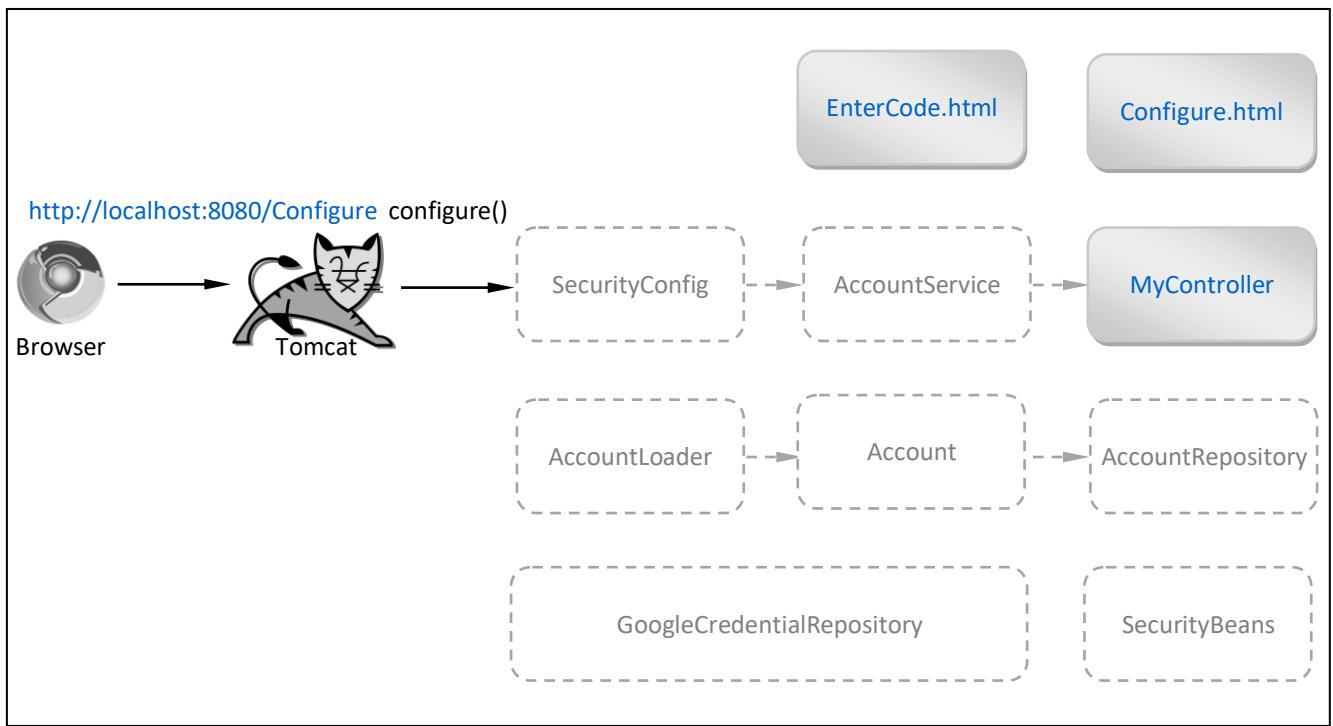
Info

[G]

- This tutorial shows how to
 - create Endpoint [/EnterCode](#) which displays HTML Page [EnterCode.html](#) where User can enter Google Temporary Code
 - which then calls Endpoint [/VerifyCode](#) which, if Code is valid, sets `account.google2faAuthenticated = true` inside DB

Application Schema

[Results]



Procedure

- >Edit Class: [MyController.java](#) (add Endpoints: EnterCode, VerifyCode)
- Edit File: [Configure.html](#) (add Button to go to EnterCode.html after configuration is over)
- Create File: [EnterCode.html](#) (for User to enter Temporary Code)

MyController.java

```
package com.ivoronline.springboot_security_2fa.controllers;

import com.ivoronline.springboot_security_2fa.entities.Account;
import com.ivoronline.springboot_security_2fa.repositories.AccountRepository;
import com.warrenstrange.googleauth.GoogleAuthenticator;
import com.warrenstrange.googleauth.GoogleAuthenticatorKey;
import com.warrenstrange.googleauth.GoogleAuthenticatorQRGenerator;
import lombok.RequiredArgsConstructor;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.security.core.userdetails.User;
import org.springframework.ui.Model;
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
@RequiredArgsConstructor
public class MyController {

    private final AccountRepository accountRepository;
    private final GoogleAuthenticator googleAuthenticator;

    //=====
    // CONFIGURE (SERVICE & MOBILE APP)
    //=====

    @RequestMapping("/Configure")
    public String configure(Model model){

        //GET USERNAME
        User user = (User) SecurityContextHolder.getContext().getAuthentication().getPrincipal();
        String username = user.getUsername();

        //CREATE NEW KEY IN DB (everytime it is called)
        GoogleAuthenticatorKey key = googleAuthenticator.createCredentials(username);

        //CONSTRUCT URL FOR QRIMAGE
        String googleURL = GoogleAuthenticatorQRGenerator.getOtpAuthURL("something", username, key);

        //ADD GOOGLE URL AS INPUT PARAMETER TO HTML PAGE
        model.addAttribute("googleURL", googleURL);

        //RETURN CONFIRMATION HTML PAGE
        return "Configure";
    }

    //=====
    // ENTER CODE
    //=====

    @RequestMapping("/EnterCode")
    public String enterCode(){
        return "EnterCode";
    }

    //=====
    // VERIFY CODE
    //=====
```

```

//=====
@ResponseBody
@RequestMapping("/VerifyCode")
public String verifyCode(@RequestParam Integer code){

    //VERIFY CODE
    User user = (User) SecurityContextHolder.getContext().getAuthentication().getPrincipal();
    String username = user.getUsername();
    Boolean validCode = googleAuthenticator.authorizeUser(username, code);

    //RETURN IF CODE IS INVALID
    if (!validCode) { return "Code is Invalid"; }

    //UPDATE ACCOUNT
    Account account = accountRepository.findByUsername(user.getUsername());
    account.google2faAuthenticated = true;
    accountRepository.save(account);

    //RETURN MESSAGE TO USER
    return "Code is Valid";
}

//=====
// Hello
//=====

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

```

Configure.html

```

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

<p>  </p>

<form method="GET" action="EnterCode">
    <p> <input type="submit" name="submit" value="Enter Temporary Code"/> </p>
</form>

```

EnterCode.html

```

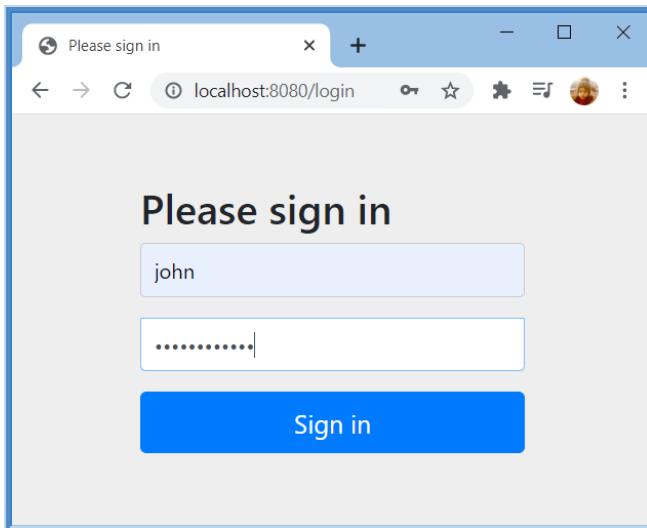
<form method="GET" action="VerifyCode">
    <p> <input type="number" name="code" placeholder="Enter Temporary Code"/> </p>
    <p> <input type="submit" name="submit"/> </p>
</form>

```

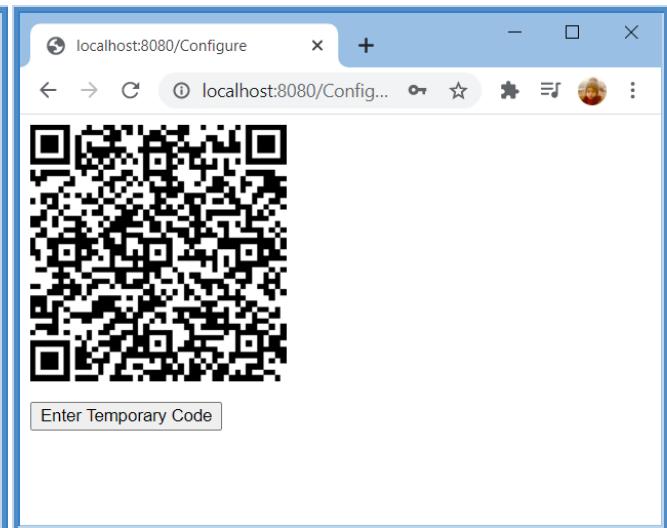
Results

- <http://localhost:8080/Configure>
- You get redirected to <http://localhost:8080/login>
 - Username: **john**
 - Password: **johnpassword**
 - Sign in
- You get redirected back to <http://localhost:8080/Configure> (shows QRCode)
 - Scan QRCode with Google Authenticator
 - It should show a Temporary Code: **267 189**
 - Enter Temporary Code
- You get redirected back to <http://localhost:8080/EnterCode>
 - Enter Temporary Code: **267 189**
 - Submit
- You get redirected to <http://localhost:8080/VerifyCode>
 - account.google2faAuthenticated = true

<http://localhost:8080/login> - john - johnpassword



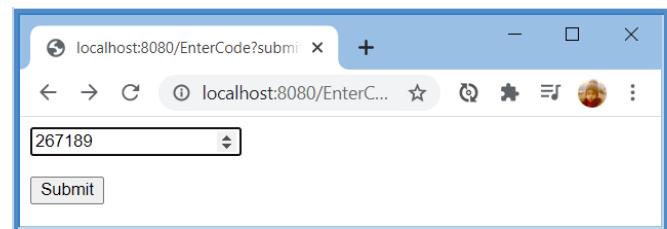
<http://localhost:8080/Configure>



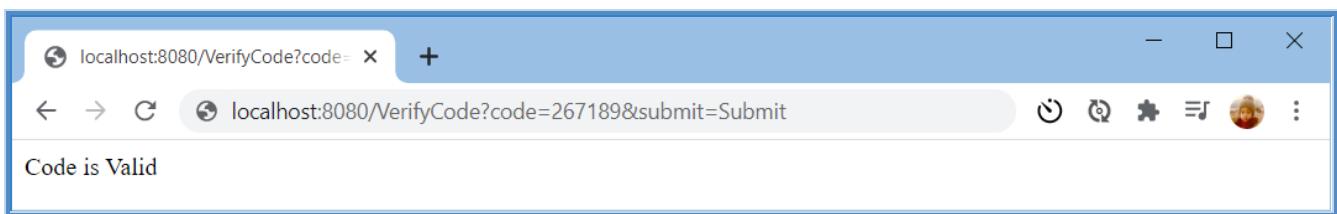
Scanning with **Google Authenticator** Mobile App



<http://localhost:8080/EnterCode>



<http://localhost:8080/VerifyCode?code=267189&submit=Submit>



Account

	id	google2fa_authenticated	google2fa_enabled	google2fa_secret_key	password	role	username
1	1	true	true	CJIFWX224S6GC4U2E52HWE43NB6ZI5D3	johnpassword	ROLE_ADMIN	john

1.10.4 Step 4: Restrict Access

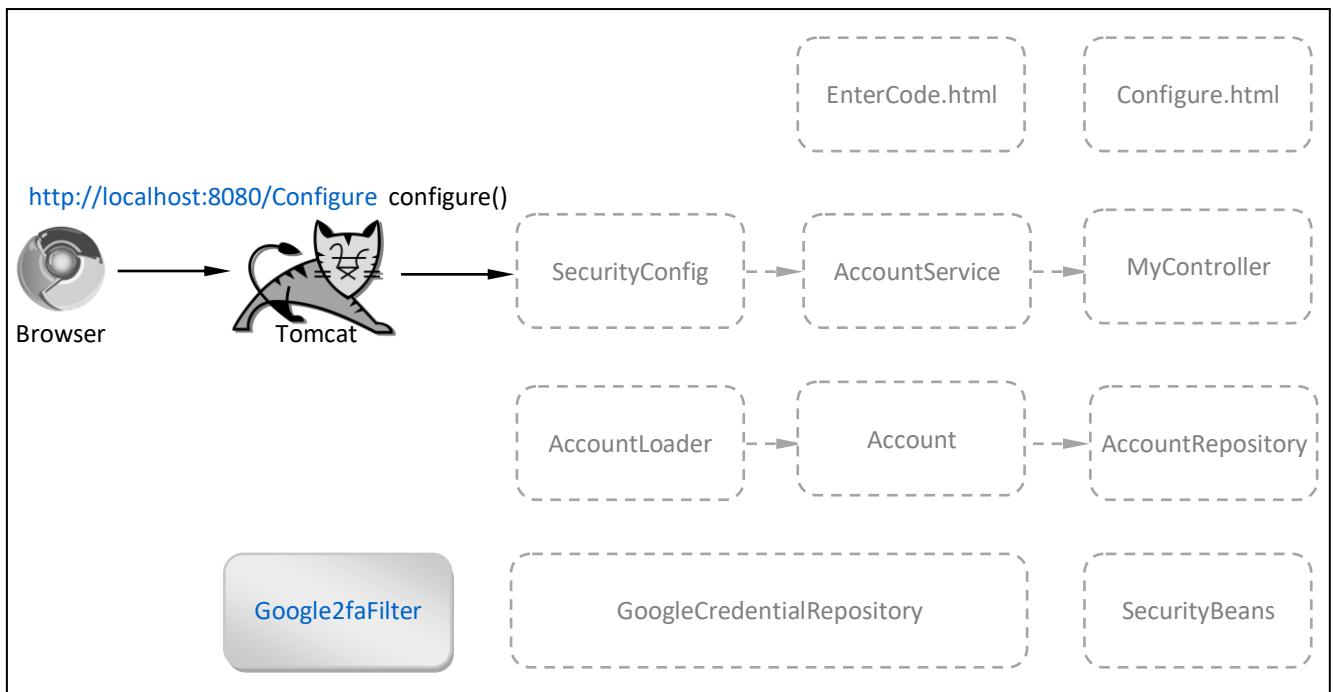
Info

[G]

- In this tutorial we will show how to restrict access to Endpoints based on 2FA Authentication.
In other words to check if User has passed 2FA Authentication before allowing him to access certain Endpoints.
More specifically, if `google2faRequired = true` then it must also be `google2faAuthenticated = true` (valid Code entered).
- Endpoints to `/Configure`, `/EnterCode` and `/VerifyCode` will be restricted behind standard Username and Password.
But they will be accessible if User is not 2FA Authenticated (since they are needed to perform 2FA Authentication).
- Endpoint `/Hello` will be restricted behind 2FA Authenticated.
User will be able to access it only when `google2faRequired = true` and `google2faAuthenticated = true` (valid Code).

Application Schema

[Results]



Procedure

- Create Class: [Google2faFilter.java](#) (inside package config)

Google2faFilter.java

```
package com.ivoronline.springboot_security_2fa.config;

import com.ivoronline.springboot_security_2fa.entities.Account;
import com.ivoronline.springboot_security_2fa.repositories.AccountRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.core.context.SecurityContextHolder;
import org.springframework.core.userdetails.User;
import org.springframework.security.web.util.matcher.AntPathRequestMatcher;
import org.springframework.security.web.util.matcher.RequestMatcher;
import org.springframework.stereotype.Component;
import org.springframework.web.filter.GenericFilterBean;
import javax.servlet.FilterChain;
import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;

@Component
public class Google2faFilter extends GenericFilterBean {

    @Autowired
    private AccountRepository accountRepository;

    @Override
    public void doFilter(ServletRequest servletRequest, ServletResponse servletResponse, FilterChain filterChain) throws IOException, ServletException {

        HttpServletRequest request = (HttpServletRequest) servletRequest;
        HttpServletResponse response = (HttpServletResponse) servletResponse;

        //ALLOW ACCESS TO FOLLOWING URLs
        RequestMatcher urlConfigure = new AntPathRequestMatcher("/Configure");
        RequestMatcher urlEnterCode = new AntPathRequestMatcher("/EnterCode");
        RequestMatcher urlVerifyCode = new AntPathRequestMatcher("/VerifyCode");
        if (urlConfigure.matches(request) || urlEnterCode.matches(request) || urlVerifyCode.matches(request)) {
            filterChain.doFilter(request, response);
            return;
        }

        //GET ACCOUNT
        User user = (User) SecurityContextHolder.getContext().getAuthentication().getPrincipal();
        String username = user.getUsername();
        Account account = accountRepository.findByUsername(username);

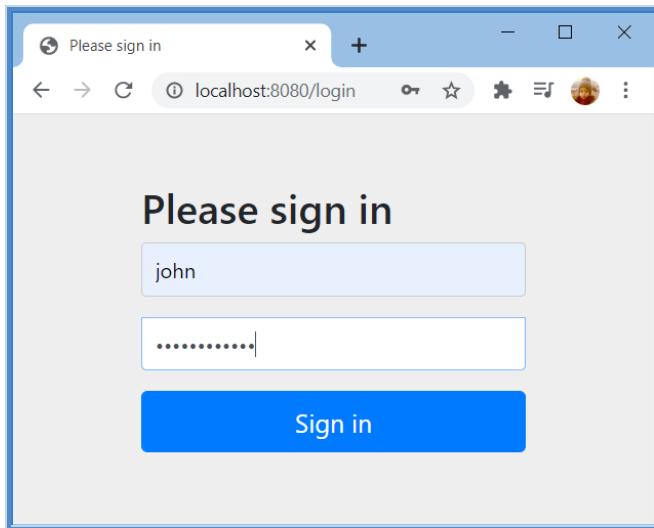
        //CHECK IF ACCOUNT SHOULD BE 2FA AUTHENTICATED
        if (account.google2faEnabled && !account.google2faAuthenticated) {
            System.out.println("Account is not 2FA Authenticated");
            return;
        }

        //ALLOW ACCESS IF ACCOUNT SHOULD NOT BE 2FA AUTHENTICATED
        filterChain.doFilter(request, response);
        return;
    }
}
```

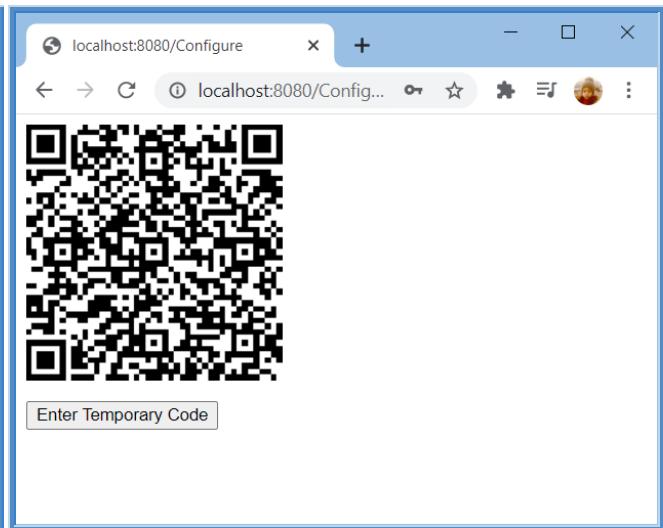
Results

- <http://localhost:8080/Configure>
- You get redirected to <http://localhost:8080/login>
 - Username: **john**
 - Password: **johnpassword**
 - Sign in
- You get redirected back to <http://localhost:8080/Configure> (shows QRCode)
 - Scan QRCode with Google Authenticator
 - It should show a Temporary Code: **267 189**
 - Enter Temporary Code
- You get redirected to <http://localhost:8080/EnterCode>
 - Enter Temporary Code: **267 189**
 - Submit
- You get redirected to <http://localhost:8080/VerifyCode> (account.google2faAuthenticated = true)
- Test Access
 - <http://localhost:8080>Hello> (allowed access google2faAuthenticated = true)
 - Set account.google2faAuthenticated = false
 - <http://localhost:8080>Hello> (restricted access google2faAuthenticated = false)
 - <http://localhost:8080/EnterCode>
 - Enter Temporary Code: **332815**
 - Submit
 - <http://localhost:8080>Hello> (allowed access google2faAuthenticated = true)

<http://localhost:8080/login> - john - johnpassword



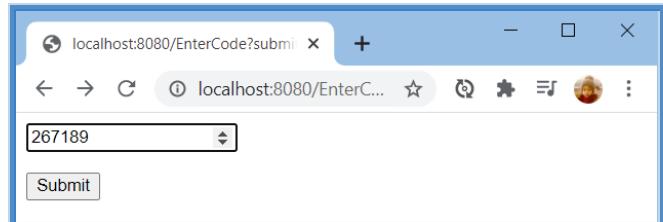
<http://localhost:8080/Configure>



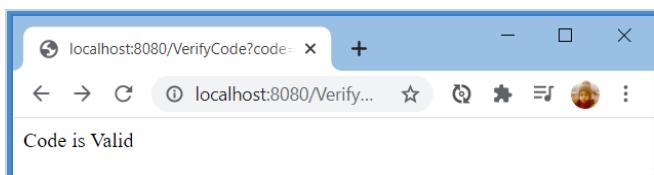
Scanning with **Google Authenticator** Mobile App



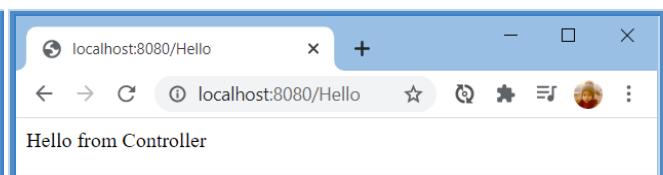
<http://localhost:8080/EnterCode>



<http://localhost:8080/VerifyCode?code=267189>



<http://localhost:8080>Hello> (Can access)



Account

	<code>id</code>	<code>google2fa_authenticated</code>	<code>google2fa_enabled</code>	<code>google2fa_secret_key</code>	<code>password</code>	<code>role</code>	<code>username</code>
1	1	• true	• true	CJIFWX224S6GC4U2E52HWE43NB6ZI5D3	johnpassword	ROLE_ADMIN	john

<http://localhost:8080>Hello> (allowed access for **true**) <http://localhost:8080>Hello> (restricted access for **false**)

The image shows two side-by-side browser windows. Both have the URL 'localhost:8080>Hello' in the address bar. The left window has a red box around the 'true' value in the database table under 'google2fa_authenticated'. The left window displays the text 'Hello from Controller' in its main content area. The right window also has a red box around the same database value but is completely blank in its content area.

<http://localhost:8080/EnterCode>

<http://localhost:8080/VerifyCode>

The image shows two browser windows. The left window is titled 'localhost:8080/EnterCode' and contains a single input field with the value '332815' and a 'Submit' button below it. The right window is titled 'localhost:8080/VerifyCode?code=' and displays the text 'Code is Valid'.

<http://localhost:8080>Hello> (allowed access for **true**)

The image shows a single browser window with the URL 'localhost:8080>Hello'. It has a red box around the 'true' value in the database table under 'google2fa_authenticated'. The window displays the text 'Hello from Controller' in its content area.

1.11 JWT (JSON Web Token)

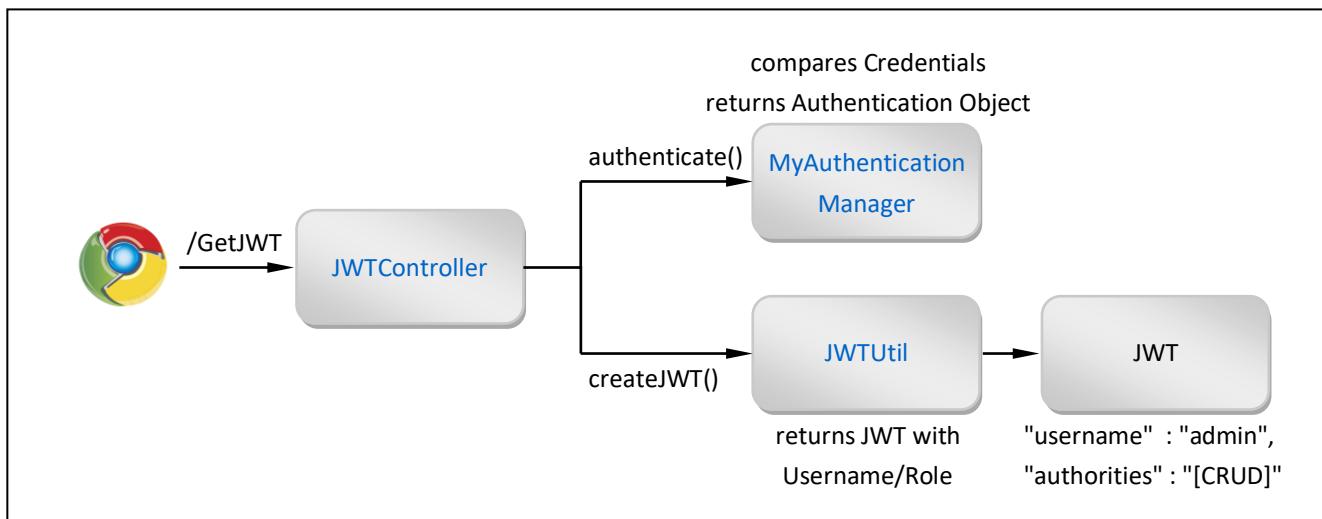
Info

[G] [R]

- This tutorial shows how to implement JWT for Spring Boot Application.
For simplicity reason Username and Role are returned as hard coded Claims.
- You can also follow subsequent tutorials to get the same result through multiple steps (recommended for first timers).
- [JWT Authorities from DB](#) tutorial shows how to use JWT in combination with Authorities from DB.
- [JWT](#) chapter contains general information about JWT (for those who are not familiar with JWT).

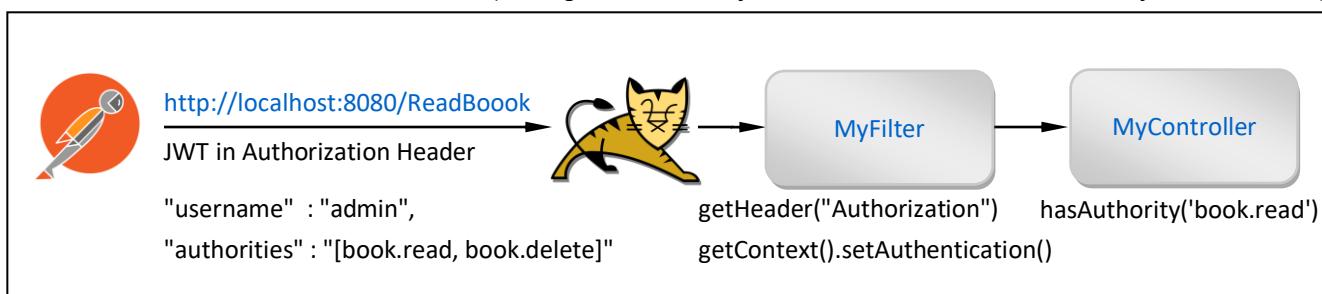
Get JWT <http://localhost:8080/GetJWT?enteredUsername=admin&enteredPassword=adminpassword>

[Results]



Send JWT

(*Filter* gets Authorities from JWT & stores Authentication Object into Context)



Spring Boot Starters

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

Overview

- `JWTController` contains single Endpoint `/GetJWT` that receives Username/Password and returns JWT
 - First it calls `MyAuthenticationManager.authenticate()` which
 - compares received Credentials with stored ones (hard coded in this example)
 - if they match returns Authentication Object with User's Role (hard coded in this example)
 - Then it calls `JWTUtil.createJWT()` to create JWT with given Username and Role
- `MyFilter` intercepts HTTP Request and
 - extracts JWT from Authorization Header
 - extracts Username and Role from JWT
 - creates Authentication Object from Username and Role
 - stores Authentication Object into Context

```
jwtUtil.extractJWTFromAuthorizationHeader()
jwtUtil.getClaims(jwt);
new UsernamePasswordAuthenticationToken()
getContext().setAuthentication(authentication)
```

Procedure

- Edit File: pom.xml (add dependencies)
- Create Package: config (inside Package security)
- Create Class: JWTUtil.java (Create JWT, extract Claims)
- Create Class: MyAuthenticationManager.java (Compare entered Credentials with stored ones)
- Create Class: WebSecurityConfig.java (Add anonymous Endpoint "/Authenticate")
- Create Class: MyFilter.java (Add anonymous Endpoint "/Authenticate")
- Create Package: controllers (inside Package security)
- Create Class: JWTController.java (/Authenticate returns JWT. Other Endpoints are for demonstration)
- Create Class: MyController.java (Access to restricted /Hello)

pom.xml

```
<dependency>
    <groupId>io.jsonwebtoken</groupId>
    <artifactId>jjwt</artifactId>
    <version>0.9.1</version>
</dependency>

<dependency>
    <groupId>javax.xml.bind</groupId>
    <artifactId>jaxb-api</artifactId>
    <version>2.3.0</version>
</dependency>
```

JWTUtil.java

```
package com.ivoronline.springboot_security_jwt.config;

import io.jsonwebtoken.Claims;
import io.jsonwebtoken.JwtBuilder;
import io.jsonwebtoken.Jwts;
import io.jsonwebtoken.SignatureAlgorithm;
import org.springframework.stereotype.Component;

import javax.crypto.spec.SecretKeySpec;
import javax.xml.bind.DatatypeConverter;
import java.security.Key;
import java.util.HashMap;
import java.util.Map;

@Component
public class JWTUtil {

    //USED TO CREATE & DECODE JWT
    public final static String SECRET_KEY = "mysecretkey";

    //=====
    // CREATE JWT
    //=====

    public String createJWT(String username, String authorities) {

        //HEADER (SPECIFY ALGORITHM)
        SignatureAlgorithm signatureAlgorithm = SignatureAlgorithm.HS256;

        //PAYLOAD (SPECIFY CLAIMS)
        Map<String, Object> customClaims = new HashMap<>();
        customClaims.put("username" , username);
        customClaims.put("authorities" , authorities);

        JwtBuilder builder = Jwts.builder()
```

```

.setClaims(customClaims) //Place them first not to override subsequent Claims
.setId("1")
.setSubject("TestJWT")
.setIssuer("ivoronline");

//SIGNATURE (SPECIFY SECRET KEY)
byte[] apiKeySecretBytes = DatatypeConverter.parseBase64Binary(SECRET_KEY);
Key signingKey = new SecretKeySpec(apiKeySecretBytes, signatureAlgorithm.getJcaName());

//EXTRACT JWT
String jwt = builder.signWith(signatureAlgorithm, signingKey).compact();

//RETURN JWT
return jwt;

}

//=====
// EXTRACT JWT FROM AUTHORIZATION HEADER
//=====

public String extractJWTFromAuthorizationHeader(String authorization) {

//GET AUTHORIZATION HEADER
if (authorization == null || !authorization.startsWith("Bearer ")) {
System.out.println("Authorization Header not found");
return null;
}

//EXTRACT JWT
String jwt = authorization.substring(7);

//RETURN JWT
return jwt;

}

//=====
// GET CLAIMS
//=====

public Claims getClaims(String jwt) {

//GET CLAIMS
Claims claims = Jwts.parser()
.setSigningKey(DatatypeConverter
.parseBase64Binary(SECRET_KEY))
.parseClaimsJws(jwt)
.getBody();

//RETURN CLAIMS
return claims;

}

//=====
// GET USERNAME
//=====

public String getUsername(String jwt) {
Claims claims = getClaims(jwt);
String username = (String) claims.get("username");
return username;
}
}

```

MyAuthenticationManager.java

```
package com.ivoronline.springboot_security_jwt.config;

import org.springframework.security.authentication.AuthenticationManager;
import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;
import org.springframework.security.core.Authentication;
import org.springframework.security.core.GrantedAuthority;
import org.springframework.security.core.authority.SimpleGrantedAuthority;
import org.springframework.stereotype.Component;
import java.util.ArrayList;
import java.util.List;

@Component
public class MyAuthenticationManager implements AuthenticationManager {

    @Override
    public Authentication authenticate(Authentication enteredAuthentication) {

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

        //GET STORED CREDENTIALS
        //Here they are hard coded (for simplicity reason)
        //Call UserDetailsService(enteredUsername) to get UserDetails with Password & Authorities from DB
        String storedUsername     = "admin";
        String storedPassword     = "adminpassword";
        String storedAuthorities = "book.read, book.delete";

        //AUTHENTICATE USER (COMPARE ENTERED AND STORED CREDENTIALS)
        if (!enteredUsername.equals(storedUsername)) { System.out.println("Username not found"); return null; }
        if (!enteredPassword.equals(storedPassword)) { System.out.println("Incorrect Password"); return null; }

        //CREATE AUTHORITIES
        String[] authoritiesArray = storedAuthorities.split(", ");
        List<GrantedAuthority> authorities = new ArrayList<GrantedAuthority>();
        for(String authority : authoritiesArray) {
            authorities.add(new SimpleGrantedAuthority(authority));
        }

        //CREATE VALIDATED AUTHENTICATION
        Authentication validatedAuth = new UsernamePasswordAuthenticationToken(enteredUsername, null,
        authorities);

        //RETURN VALIDATES AUTHENTICATION
        return validatedAuth;
    }
}
```

WebSecurityConfig.java

```
package com.ivoronline.springboot_security_jwt.config;

import org.springframework.beans.factory.annotation.Autowired;
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.EnableWebSecurity;
import org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;
import org.springframework.security.config.http.SessionCreationPolicy;
import org.springframework.security.web.authentication.UsernamePasswordAuthenticationFilter;

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

    @Autowired private MyFilter myFilter;

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

        //ANONYMOUS ACCESS
        httpSecurity.authorizeRequests().antMatchers("/GetJWT").permitAll(); //To get JWT
        httpSecurity.authorizeRequests().antMatchers("/CreateJWT").permitAll(); //To get JWT
        httpSecurity.authorizeRequests().antMatchers("/GetClaims").permitAll(); //For Testing
        httpSecurity.authorizeRequests().antMatchers("/GetUsername").permitAll(); //For Testing

        //OTHER CONFIGURATION
        httpSecurity.csrf().disable(); //Enables POST
        httpSecurity.authorizeRequests().anyRequest().authenticated(); //Authenticated
        httpSecurity.addFilterBefore(myFilter, UsernamePasswordAuthenticationFilter.class); //Add Filter
        httpSecurity.sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS); //No Session

    }

}
```

MyFilter.java

```
package com.ivoronline.springboot_security_jwt.config;

import java.io.IOException;
import java.util.ArrayList;
import java.util.List;
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 io.jsonwebtoken.Claims;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;
import org.springframework.security.core.Authentication;
import org.springframework.security.core.GrantedAuthority;
import org.springframework.security.core.authority.SimpleGrantedAuthority;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.stereotype.Component;

@Component
public class MyFilter implements Filter {

    @Autowired JWTUtil jwtUtil;
```

```

//=====
// DO FILTER
//=====

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

    //CAST TO GET ACCESS TO HEADERS
    HttpServletRequest httpRequest = (HttpServletRequest) request;

    //GET AUTHORIZATION HEADER
    String authorizationHeader = httpRequest.getHeader("Authorization");

    //IF AUTHORIZATION HEADER EXISTS USE JWT TO PUT AUTHENTICATION OBJECT INTO CONTEXT
    if(authorizationHeader != null) { addAuthenticationObjectIntoContext(authorizationHeader); }

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

}

//=====
// ADD AUTHENTICATION OBJECT INTO CONTEXT
//=====

private void addAuthenticationObjectIntoContext(String authorizationHeader) {

    //EXTRACT JWT FROM AUTHORIZATION HEADER
    String jwt = jwtUtil.extractJWTFromAuthorizationHeader(authorizationHeader);

    //GET CLAIMS
    Claims claims      = jwtUtil.getClaims(jwt);
    String username     = (String) claims.get("username");
    String authoritiesJWT = (String) claims.get("authorities"); //"[book.read, book.delete]"

    //CREATE AUTHORITIES
    String authoritiesString = authoritiesJWT.replace("[", "").replace("]", "").replace(" ", "");
    String[] authoritiesArray = authoritiesString.split(",");
    List<GrantedAuthority> authorities = new ArrayList<GrantedAuthority>();
    for(String authority : authoritiesArray) {
        authorities.add(new SimpleGrantedAuthority(authority));
    }

    //AUTHENTICATE
    Authentication authentication = new UsernamePasswordAuthenticationToken(username, null, authorities);

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

}
}

```

JWTController.java

```
package com.ivorononline.springboot_security_jwt.controllers;

import com.ivorononline.springboot_security_jwt.config.JWTUtil;
import com.ivorononline.springboot_security_jwt.config.MyAuthenticationManager;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;
import org.springframework.security.core.Authentication;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.*;

@Controller
public class JWTController {

    @Autowired
    JWTUtil jwtUtil;
    MyAuthenticationManager myAuthenticationManager;

    //=====
    // GET JWT
    //=====

    @ResponseBody
    @RequestMapping("/GetJWT")
    public String getJWT(@RequestParam String enteredUsername, @RequestParam String enteredPassword) {

        //AUTHENTICATE (COMPARE ENTERED AND STORED CREDENTIALS)
        Authentication enteredAuth = new UsernamePasswordAuthenticationToken(enteredUsername, enteredPassword);
        Authentication returnedAuth = myAuthenticationManager.authenticate(enteredAuth);

        //CHECK RESULT OF AUTHENTICATION
        if(returnedAuth == null) { return "User is NOT Authenticated"; }

        //CREATE JWT
        String username = (String) returnedAuth.getPrincipal();
        String authorities = (String) returnedAuth.getAuthorities().toString(); //"[CREATE, READ]"
        String jwt = jwtUtil.createJWT(username, authorities);

        //RETURN JWT
        return jwt;
    }
}
```

MyController.java

```
package com.ivorononline.springboot_security_jwt.controllers;

import org.springframework.security.access.prepost.PreAuthorize;
import org.springframework.web.bind.annotation.*;
import org.springframework.stereotype.Controller;

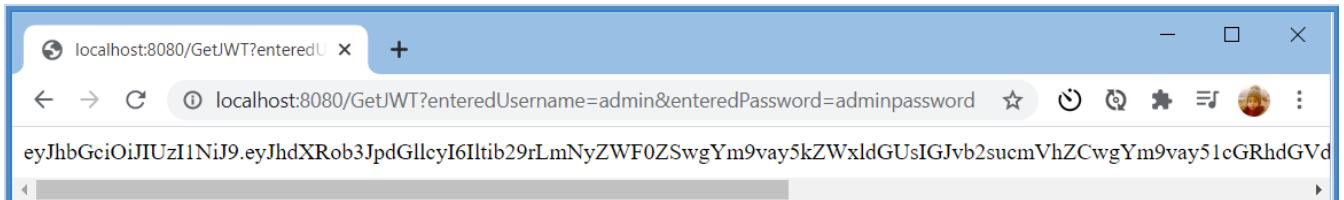
@Controller
public class MyController {

    @ResponseBody
    @RequestMapping("/ReadBook")
    @PreAuthorize("hasAuthority('book.read')")
    public String hello() {
        return "ADMIN can read books";
    }
}
```

Results

Get JWT

<http://localhost:8080/GetJWT?enteredUsername=admin&enteredPassword=adminpassword>



<https://jwt.io/#debugger>

(Decoded JWT)

```
{  
    "username" : "admin"  
    "authorities" : "[book.read, book.delete]",  
    "iss" : "ivoronline",  
    "sub" : "TestJWT",  
    "jti" : "1"  
}
```

Send JWT (With Postman in Authentication Header)

GET or POST

<http://localhost:8080/ReadBook>

Headers

(add Key-Value)

Authorization: Bearer
eyJhbGciOiJIUzI1NiJ9.eyJdWIIoIJUZXN0SldUiwi...dXNlc...g7CcreAHQvSR1JjQJqTBvT3eGcpz9kcuV0kgUFREiIc

Postman

The screenshot shows the Postman application interface. A GET request is being made to <http://localhost:8080/ReadBook>. In the Headers tab, there are two entries: 'Connection' with the value 'keep-alive' and 'Authorization' with the value 'Bearer eyJhbGciOiJIUzI1NiJ9.eyJdWIIoIJUZXN0SldUiwi...dXNlc...g7CcreAHQvSR1JjQJqTBvT3eGcpz9kcuV0kgUFREiIc'. The response status is 200 OK, time is 38 ms, and size is 366 B.

Application Structure

The screenshot shows the IntelliJ IDEA interface with the following details:

- Project Tree:** The project is named "springboot_security_jwt". It contains a ".idea" folder, a ".mvn" folder, a "src" directory with "main" and "java" sub-directories. The "java" directory contains packages "com.ivoronline.springboot_security.jwt" (with sub-folders "config", "controllers", and "WebSecurityConfig") and "com.springframework.boot.SpringApplication".
- Code Editor:** The file "JWTController.java" is open. The code implements a REST endpoint to generate a JWT token. It uses annotations like @ResponseBody and @RequestMapping. It interacts with "JWTUtil", "MyAuthenticationManager", and "MyFilter" classes.
- Toolbars and Status Bar:** Standard IntelliJ toolbars are visible at the top. The status bar at the bottom shows the current time (14:14), encoding (CRLF), and file system information (Event Log).

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>io.jsonwebtoken</groupId>
        <artifactId>jjwt</artifactId>
        <version>0.9.1</version>
    </dependency>

    <dependency>
        <groupId>javax.xml.bind</groupId>
        <artifactId>jaxb-api</artifactId>
        <version>2.3.0</version>
    </dependency>

</dependencies>
```

1.11.1 Step 1 - Get Token

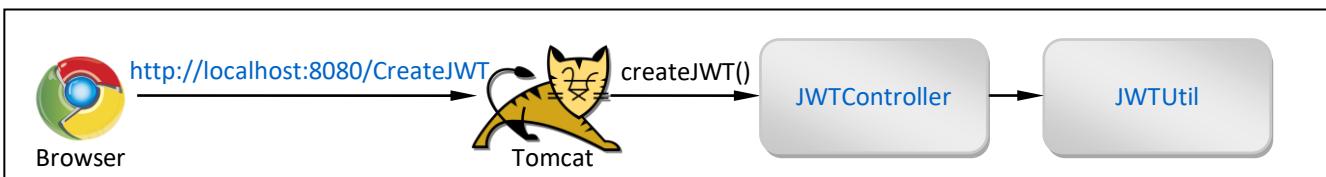
Info

[G]

- This tutorial shows how to make Endpoint that returns JWT.
- Username and Role that will go into JWT will be hard coded and no Authentication will be performed (HTTP Request will not contain Credentials that could be compared to stored Credentials).

Application Schema

[Results]



Spring Boot Starters

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

Overview

JWTController.java

```
@ResponseBody  
@RequestMapping("/CreateJWT")  
public String createJWT() {  
    return JWTUtil.createJWT("admin", "[book.create, book.delete]");  
}
```

JWT

```
{  
    "username" : "myuser",  
    "authorities" : "[book.create, book.delete]"  
}
```

<http://localhost:8080/CreateJWT>

```
eyJhbGciOiJIUzI1NiJ9.eyJqdGkiOiIxIiwiaXNzIjoiaXZvcm9ubGluZSJzInN1YiI6IlRlc3RKV1QifQ.GZkuBtau-7uEJb7V1-  
1mGu8q3YmjPzYCok_qfHHhP9Y
```

Procedure

- [Create Project:](#) `springboot_security_jwt` (add Spring Boot Starters from the table)
- [Edit File:](#) `pom.xml` (manually add JWT dependency)
- [Create Package:](#) `config` (inside main package)
 - [Create Class:](#) `JWTUtil.java` (inside package controllers)
- [Create Package:](#) `controllers` (inside main package)
 - [Create Class:](#) `JWTController.java` (inside package controllers)

pom.xml

```
<dependency>
    <groupId>io.jsonwebtoken</groupId>
    <artifactId>jjwt</artifactId>
    <version>0.9.1</version>
</dependency>

<dependency>
    <groupId>javax.xml.bind</groupId>
    <artifactId>jaxb-api</artifactId>
    <version>2.3.0</version>
</dependency>
```

JWTController.java

```
package com.ivoronline.springboot_security_jwt.controllers;

import com.ivoronline.springboot_security_jwt.config.JWTUtil;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.ResponseBody;

@Controller
public class JWTController {

    @ResponseBody
    @RequestMapping("/CreateJWT")
    public String createJWT() {
        String jwt = JWTUtil.createJWT("admin", "[book.create, book.delete]");
        return jwt;
    }
}
```

```

package com.ivoronline.springboot_security_jwt.config;

import io.jsonwebtoken.JwtBuilder;
import io.jsonwebtoken.Jwts;
import io.jsonwebtoken.SignatureAlgorithm;
import javax.crypto.spec.SecretKeySpec;
import javax.xml.bind.DatatypeConverter;
import java.security.Key;
import java.util.HashMap;
import java.util.Map;

public class JWTUtil {

    //USED TO CREATE & DECODE JWT
    public final static String SECRET_KEY = "mysecretkey";

    //=====
    // CREATE JWT
    //=====

    public static String createJWT(String username, String authorities) {

        //HEADER (SPECIFY ALGORITHM)
        SignatureAlgorithm signatureAlgorithm = SignatureAlgorithm.HS256;

        //PAYLOAD (SPECIFY CLAIMS)
        Map<String, Object> customClaims = new HashMap<>();
        customClaims.put("username" , username);
        customClaims.put("authorities", authorities);

        JwtBuilder builder = Jwts.builder()
            .setClaims (customClaims) //Place them first not to override subsequent Claims
            .setId      ("1")
            .setSubject("TestJWT")
            .setIssuer  ("ivoronline");

        //SIGNATURE (SPECIFY SECRET KEY)
        byte[] apiKeySecretBytes = DatatypeConverter.parseBase64Binary(SECRET_KEY);
        Key     signingKey       = new SecretKeySpec(apiKeySecretBytes, signatureAlgorithm.getJcaName());

        //GENERATE JWT
        String jwt = builder.signWith(signatureAlgorithm, signingKey).compact();
        return jwt;
    }
}

```

Results

<http://localhost:8080/CreateJWT>



Encoded JWT

```
eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJUZXN0SldUIiwiaXNzIjoiaXZvcm9ubGluZSIssImF1dGhvcmI0aWVzIjoiW2Jvb2suY3J1YXR1LCBi b29rLmRlbGV0ZV0iLCJqdGkiOiIxIiwidXNlcm5hbWUiOiJhZG1pbij9.jQhRDTMA_nsHhH70sShoGiWp6rmeFWqkcIG14LaFP0o
```

Decoded JWT

```
{
  "sub": "TestJWT",
  "iss": "ivoronline",
  "authorities": "[book.create, book.delete]",
  "jti": "1",
  "username": "admin"
}
```

<https://jwt.io>

A screenshot of the jwt.io debugger interface. The top navigation bar shows "JSON Web Tokens - jwt.io" and the URL "jwt.io/#debugger". The page features a logo with the word "JUUT" and links for "Debugger", "Libraries", "Introduction", "Ask", and "Get a T-shirt!". On the right, there are sections for "Encoded" and "Decoded" tokens.

Encoded (PASTE A TOKEN HERE):
eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJUZXN0SldUIiwiaXNzIjoiaXZvcm9ubGluZSIssImF1dGhvcmI0aWVzIjoiW2Jvb2suY3J1YXR1LCBi b29rLmRlbGV0ZV0iLCJqdGkiOiIxIiwidXNlcm5hbWUiOiJhZG1pbij9.jQhRDTMA_nsHhH70sShoGiWp6rmeFWqkcIG14LaFP0o

Decoded (EDIT THE PAYLOAD AND SECRET):

HEADER: ALGORITHM & TOKEN TYPE

```
{
  "alg": "HS256"
}
```

PAYOUT: DATA

```
{
  "sub": "TestJWT",
  "iss": "ivoronline",
  "authorities": "[book.create, book.delete]",
  "jti": "1",
  "username": "admin"
}
```

VERIFY SIGNATURE

```
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  your-256-bit-secret
)
```

secret base64 encoded

1.11.2 Step 2 - Send Token - As Request Parameter - Get Claims

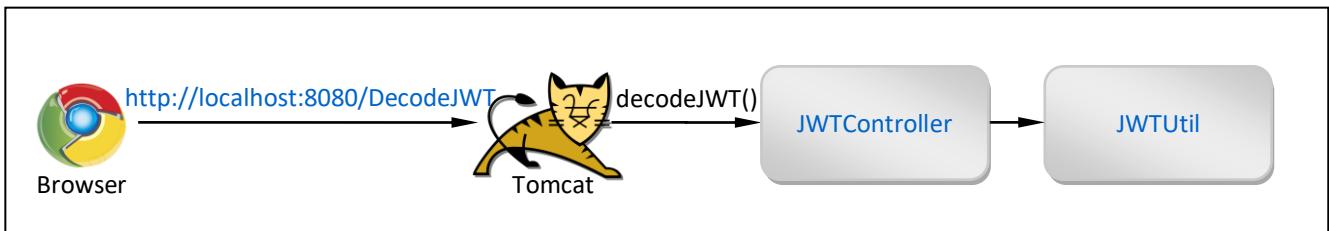
Info

[G]

- This tutorial shows how to send JWT as HTTP Request Parameter.
- [decodeJWT\(\)](#) Endpoint will then decode JWT and return extracted Claims.

Application Schema

[Results]



Overview

JWTController.java

```
@ResponseBody  
@RequestMapping("/DecodeJWT")  
public Claims decodeJWT(@RequestParam String jwt) {  
    Claims claims = jwtUtil.decodeJWT(jwt);  
    return claims;  
}
```

http://localhost:8080/DecodeJWT?jwt=eyJhbGciOiJIUzI1NiJ9eyJqdGkiOiIxliwiaXNzIjoiZXvcm9ubGluZSIsInN1Yil6IlRlc3RKV1QifQ.GZkuBtau-7uEJb7V1-1mGu8q3YmjPzYCok_qfHHhP9Y

```
{  
    "username" : "admin",  
    "authorities" : "[book.create, book.delete]",  
}
```

Procedure

- >Edit Class: [JWTUtil.java](#) (Add decodeJWT())
- Edit Class: [JWTController.java](#) (Add /DecodeJWT)

JWTController.java

```
package com.ivoronline.springboot_security_jwt.controllers;

import com.ivoronline.springboot_security_jwt.config.JWTUtil;
import io.jsonwebtoken.Claims;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.ResponseBody;

@Controller
public class JWTController {

    @ResponseBody
    @RequestMapping("/CreateJWT")
    public String createJWT() {
        String jwt = JWTUtil.createJWT("admin", "[book.create, book.delete]");
        return jwt;
    }

    @ResponseBody
    @RequestMapping("/DecodeJWT")
    public Claims decodeJWT(@RequestParam String jwt) {

        //GET CLAIMS
        Claims claims = JWTUtil.decodeJWT(jwt);

        //RETURN CLAIMS
        return claims;
    }
}
```

JWTUtil.java

```
package com.ivoronline.springboot_security_jwt.config;

import io.jsonwebtoken.Claims;
import io.jsonwebtoken.JwtBuilder;
import io.jsonwebtoken.Jwts;
import io.jsonwebtoken.SignatureAlgorithm;
import javax.crypto.spec.SecretKeySpec;
import javax.xml.bind.DatatypeConverter;
import java.security.Key;
import java.util.HashMap;
import java.util.Map;

public class JWTUtil {

    //USED TO CREATE & DECODE JWT
    public final static String SECRET_KEY = "mysecretkey";

    //=====
    // CREATE JWT
    //=====

    public static String createJWT(String username, String authorities) {

        //HEADER (SPECIFY ALGORITHM)
        SignatureAlgorithm signatureAlgorithm = SignatureAlgorithm.HS256;

        //PAYLOAD (SPECIFY CLAIMS)
        Map<String, Object> customClaims = new HashMap<>();
        customClaims.put("username", username);
        customClaims.put("authorities", authorities);

        JwtBuilder builder = Jwts.builder()
            .setClaims(customClaims) //Place them first not to override subsequent Claims
            .setId("1")
            .setSubject("TestJWT")
            .setIssuer("ivoronline");

        //SIGNATURE (SPECIFY SECRET KEY)
        byte[] apiKeySecretBytes = DatatypeConverter.parseBase64Binary(SECRET_KEY);
        Key signingKey = new SecretKeySpec(apiKeySecretBytes, signatureAlgorithm.getJcaName());

        //GENERATE JWT
        String jwt = builder.signWith(signatureAlgorithm, signingKey).compact();
        return jwt;
    }

    //=====
    // DECODE JWT
    //=====

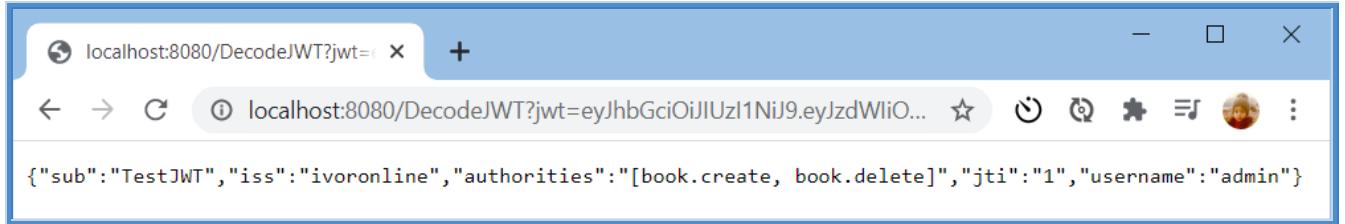
    public static Claims decodeJWT(String jwt) {

        //GET CLAIMS
        Claims claims = Jwts.parser()
            .setSigningKey(DatatypeConverter.parseBase64Binary(SECRET_KEY))
            .parseClaimsJws(jwt).getBody();

        //RETURN CLAIMS
        return claims;
    }
}
```

Results

http://localhost:8080/DecodeJWT?jwt=eyJhbGciOiJIUzI1NiJ9eyJqdGkiOiIxliwiaXNzIjoiSXzcm9ubGluZSIsInN1Yil6IlRlc3RKV1QifQ.GZkuBtau-7uEJb7V1-1mGu8q3YmjPzYCok_qfHHhP9Y



Extracted Claims

```
{  
    "sub" : "TestJWT",  
    "iss" : "ivoronline",  
    "authorities" : "[book.create, book.delete]",  
    "jti" : "1",  
    "username" : "admin"  
}
```

1.11.3 Step 3 - Send Token - In Authorization Header - Get Claims

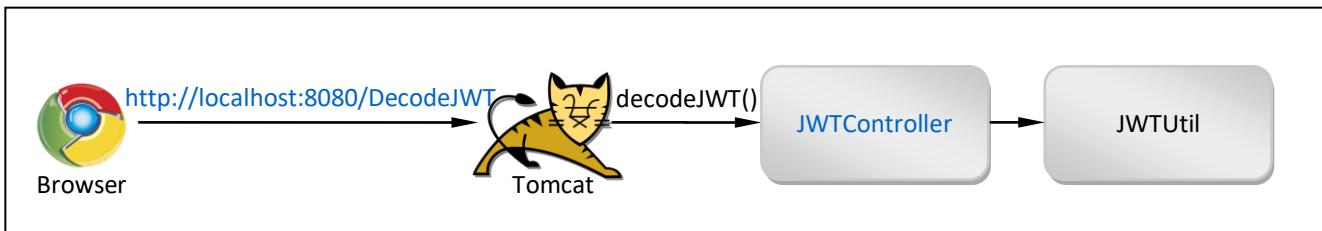
Info

[G]

- This tutorial shows how to use Postman to send JWT through Authorization Header.
- For that purpose we will need to change `decodeJWT()` to get JWT from Authorization Header.

Application Schema

[Results]



Overview

JWTController.java

```
@ResponseBody  
@RequestMapping("/DecodeJWT")  
public Claims decodeJWT(@RequestHeader("Authorization") String authorization) {  
    String jwt = jwtUtil.extractJWTFromAuthorizationHeader(authorization);  
    Claims claims = jwtUtil.decodeJWT(jwt);  
    return claims;  
}
```

POST <http://localhost:8080/CreateJWT>

(JWT in Authorization Header)

```
{  
    "username" : "admin",  
    "authorities" : "[book.create, book.delete]",  
}
```

Procedure

- Edit Class: [JWTController.java](#) (edit decodeJWT() to get JWT from Authorization Header)

JWTController.java

```
package com.ivoronline.springboot_security_jwt.controllers;

import com.ivoronline.springboot_security_jwt.config.JWTUtil;
import io.jsonwebtoken.Claims;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestHeader;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.ResponseBody;

@Controller
public class JWTController {

    @ResponseBody
    @RequestMapping("/CreateJWT")
    public String createJWT() {
        String jwt = JWTUtil.createJWT("admin", "[book.create, book.delete]");
        return jwt;
    }

    @ResponseBody
    @RequestMapping("/DecodeJWT")
    public Claims decodeJWT(@RequestHeader("Authorization") String authorization) {

        //GET AUTHORIZATION HEADER
        if (authorization == null || !authorization.startsWith("Bearer ")) {
            System.out.println("Authorization Header not found");
            return null;
        }

        //GET JWT
        String jwt = authorization.substring(7);

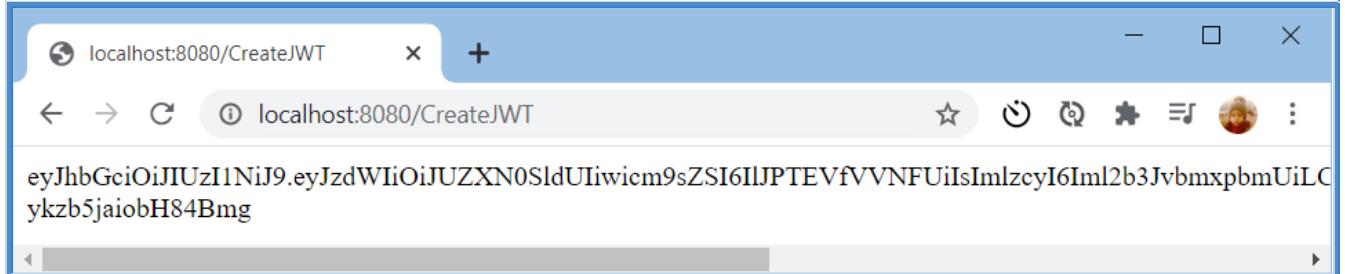
        //GET CLAIMS
        Claims claims = JWTUtil.decodeJWT(jwt);

        //RETURN CLAIMS
        return claims;
    }
}
```

Results

- <http://localhost:8080/CreateJWT>
- Start Postman
 - POST: <http://localhost:8080/DecodeJWT>
 - Headers: (copy from below)
 - Send

<http://localhost:8080/CreateJWT>



Headers

(add Key-Value)

```
Authorization: Bearer  
eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJUZXN0SldUIiwicm9sZSI6IlJPTEVfVVNFUiIsImlzcyI6Iml2b3JvbmxpbmUiLC  
ykb5jaiobH84Bmg
```

Postman

A screenshot of the Postman application interface. The top navigation bar shows "Postman" and various menu options. The main workspace shows a collection named "JWT / http://localhost:8080/DecodeJWT". A GET request is selected with the URL "http://localhost:8080/DecodeJWT". In the "Headers" tab, there is an "Authorization" header set to "Bearer" followed by the previously copied JWT token. The "Body" tab shows the decoded JSON response, which includes fields like "sub": "TestJWT", "iss": "ivoronline", "authorities": "[book.create, book.delete]", "jti": "1", and "username": "admin". The status bar at the bottom indicates a 200 OK response with a size of 272 B.

1.11.4 Step 4 - Send Token - In Authorization Header - Get Username

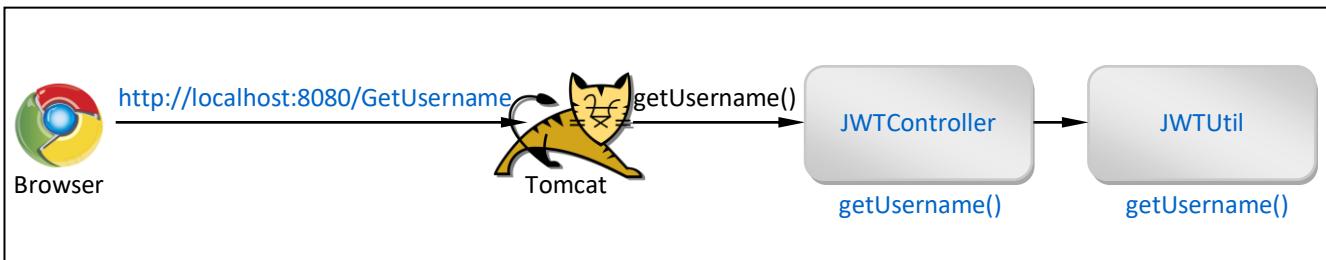
Info

[G]

- This tutorial shows how to get Username from JWT.

Application Schema

[Results]



Overview

JWTController.java

```
@ResponseBody  
 @RequestMapping("/GetUsername")  
 public String getUsername(@RequestHeader("Authorization") String authorization) {  
     String jwt = jwtUtil.extractJWTFromAuthorizationHeader(authorization);  
     return jwtUtil.getUsername(jwt);  
 }
```

POST <http://localhost:8080/GetUsername>

(JWT in Authorization Header)

admin

Procedure

- >Edit Class: [JWTController.java](#) (Add Method getUsername())
- Edit Class: [JWTUtil.java](#) (Add Method getUsername())

JWTController.java

```
package com.ivoronline.springboot_security_jwt.controllers;

import com.ivoronline.springboot_security_jwt.config.JWTUtil;
import io.jsonwebtoken.Claims;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestHeader;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.ResponseBody;

@Controller
public class JWTController {

    @Autowired
    JWTUtil jwtUtil;

    //=====
    // CREATE JWT
    //=====

    @ResponseBody
    @RequestMapping("/CreateJWT")
    public String createJWT() {

        //CREATE JWT
        String jwt = jwtUtil.createJWT("admin", "[book.create, book.delete]");

        //RETURN JWT
        return jwt;
    }

    //=====
    // GET CLAIMS
    //=====

    @ResponseBody
    @RequestMapping("/GetClaims")
    public Claims getClaims(@RequestHeader("Authorization") String authorization) {

        //EXTRACT JWT FROM AUTHORIZATION HEADER
        String jwt = jwtUtil.extractJWTFromAuthorizationHeader(authorization);

        //GET CLAIMS
        Claims claims = jwtUtil.getClaims(jwt);

        //RETURN CLAIMS
        return claims;
    }

    //=====
    // GET USERNAME
    //=====

    @ResponseBody
    @RequestMapping("/GetUsername")
    public String getUsername(@RequestHeader("Authorization") String authorization) {

        //EXTRACT JWT FROM AUTHORIZATION HEADER
        String jwt = jwtUtil.extractJWTFromAuthorizationHeader(authorization);
```

```

    //GET USERNAME
    String username = jwtUtil.getUsername(jwt);

    //RETURN USERNAME
    return username;

}

}

```

JWTUtil.java

```

package com.ivoronline.springboot_security_jwt.config;

import io.jsonwebtoken.Claims;
import io.jsonwebtoken.JwtBuilder;
import io.jsonwebtoken.Jwts;
import io.jsonwebtoken.SignatureAlgorithm;
import org.springframework.stereotype.Component;

import javax.crypto.spec.SecretKeySpec;
import javax.xml.bind.DatatypeConverter;
import java.security.Key;
import java.util.HashMap;
import java.util.Map;

@Component
public class JWTUtil {

    //USED TO CREATE & DECODE JWT
    public final static String SECRET_KEY = "mysecretkey";

    //=====
    // CREATE JWT
    //=====

    public String createJWT(String username, String authorities) {

        //HEADER (SPECIFY ALGORITHM)
        SignatureAlgorithm signatureAlgorithm = SignatureAlgorithm.HS256;

        //PAYLOAD (SPECIFY CLAIMS)
        Map<String, Object> customClaims = new HashMap<>();
        customClaims.put("username", username);
        customClaims.put("authorities", authorities);

        JwtBuilder builder = Jwts.builder()
            .setClaims(customClaims) //Place them first not to override subsequent Claims
            .setId("1")
            .setSubject("TestJWT")
            .setIssuer("ivoronline");

        //SIGNATURE (SPECIFY SECRET KEY)
        byte[] apiKeySecretBytes = DatatypeConverter.parseBase64Binary(SECRET_KEY);
        Key signingKey = new SecretKeySpec(apiKeySecretBytes, signatureAlgorithm.getJcaName());

        //EXTRACT JWT
        String jwt = builder.signWith(signatureAlgorithm, signingKey).compact();

        //RETURN JWT
        return jwt;

    }
    //=====
}

```

```

// EXTRACT JWT FROM AUTHORIZATION HEADER
//=====
public String extractJWTFromAuthorizationHeader(String authorization) {

    //GET AUTHORIZATION HEADER
    if (authorization == null || !authorization.startsWith("Bearer ")) {
        System.out.println("Authorization Header not found");
        return null;
    }

    //EXTRACT JWT
    String jwt = authorization.substring(7);

    //RETURN JWT
    return jwt;

}

//=====
// GET CLAIMS
//=====
public Claims getClaims(String jwt) {

    //GET CLAIMS
    Claims claims = Jwts.parser()
        .setSigningKey(DatatypeConverter
        .parseBase64Binary(SECRET_KEY))
        .parseClaimsJws(jwt)
        .getBody();

    //RETURN CLAIMS
    return claims;

}

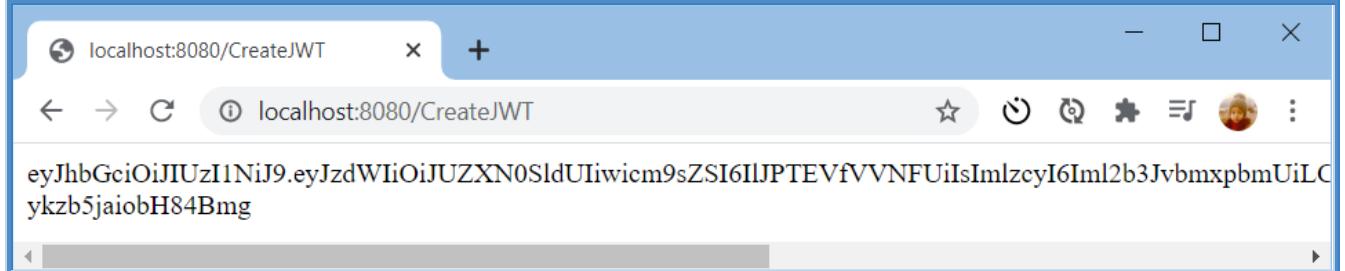
//=====
// GET USERNAME
//=====
public String getUsername(String jwt) {
    Claims claims = getClaims(jwt);
    String username = (String) claims.get("username");
    return username;
}

```

Results

- <http://localhost:8080/CreateJWT>
- Start Postman
 - POST: <http://localhost:8080/GetUsername>
 - Headers: (copy from below)
 - Send

<http://localhost:8080/CreateJWT>



Headers

(add Key-Value)

```
Authorize: Bearer  
eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJUZXN0SldUIiwicm9sZSI6IlJPTEVfVVNFUiIsImlzcyI6Iml2b3JvbmxpbmUiLC  
ykbzb5jaiobH84Bmg
```

Postman

A screenshot of the Postman application interface. The top navigation bar includes "File", "Edit", "View", "Help", "Home", "Workspaces", "Reports", "Explore", and a search bar. The main workspace shows a collection named "JWT / http://localhost:8080/ReadBook". A GET request is selected for the URL "http://localhost:8080/GetUsername". The "Headers" tab is active, showing the following configuration:

Header	Value
Accept-Encoding	gzip, deflate, br
Connection	keep-alive
Authorization	Bearer eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJUZXN0SldUIiwiaXN zIjoiaXvcm9ubGluZSlsImF1dGhvcml0aWVzIjoiW2Jvb 2sucmVhZCwgYm9vay5kZWxldGVdiwianRpIjoiMSlsIn VzZXJuYW1ljojYWRtaW4ifQ.QE6A6VAi2fLVQzVGom ajQpmXC5GRJ25iVPhv-nZGic

The "Body" tab shows the response body, which contains the value "admin". The status bar at the bottom indicates "Status: 200 OK Time: 179 ms Size: 168 B".

1.11.5 Step 5 - Filter

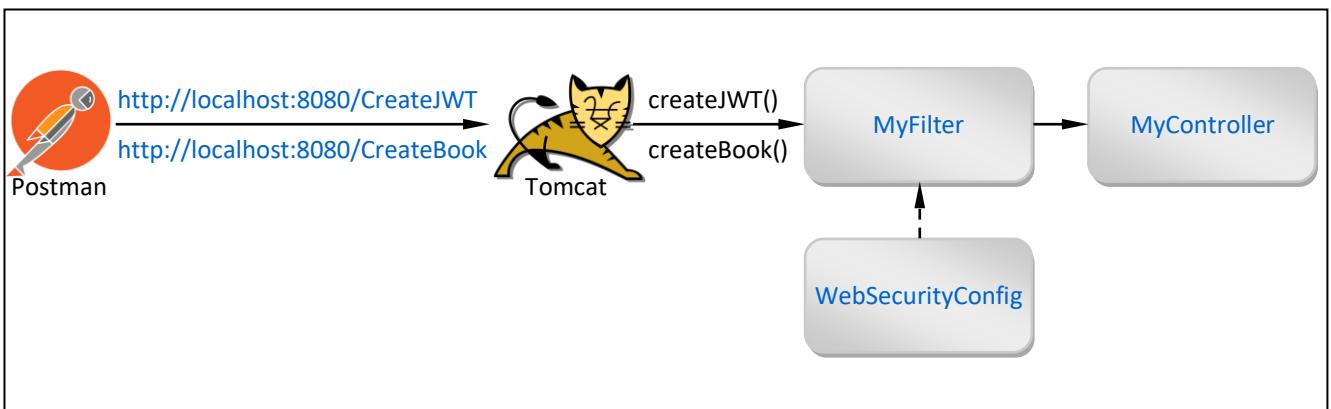
Info

[G]

- In this tutorial we will add **Filter** that for each Request creates **Authentication** Object with Username and Authorities.
- Authentication** Object is then stored into **Context** so that it can be used to control access to **MyController** Endpoints.
- Username and Role are contained inside JWT so there is no need to go into DB to get Roles/Authorities for the User.

Application Schema

[Results]



Overview

MyFilter.java

```
@Component
public class MyFilter implements Filter {

    @Override
    public void doFilter(ServletRequest request, ServletResponse response, FilterChain filterchain) {

        //GET AUTHORIZATION HEADER
        String authorizationHeader = httpRequest.getHeader("Authorization");

        if(authorizationHeader != null) {

            //EXTRACT JWT FROM AUTHORIZATION HEADER
        }
    }
}
```

Procedure

- Edit File: pom.xml (Add Security dependency)
- Inside Package: controllers
 - Create Class: MyController.java (Add restricted Endpoint "/Hello")
- Inside Package: config
 - Create Class: MyFilter.java (Store Authentication into Context)
 - Create Class: WebSecurityConfig.java (Restrict access to Endpoint "/Hello")

pom.xml

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

MyController.java

```
package com.ivoronline.springboot_security_jwt.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;

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

@Controller
public class MyController {

    @ResponseBody
    @PreAuthorize("hasAuthority('book.create')")
    @RequestMapping("/CreateBook")
    public String createBook() {
        return "Only ADMIN can create Book";
    }
}
```

MyFilter.java

```
package com.ivoronline.springboot_demo_jwtauthoritiesfromdb.security.jwt;

import io.jsonwebtoken.Claims;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;
import org.springframework.security.core.Authentication;
import org.springframework.security.core.GrantedAuthority;
import org.springframework.security.core.authority.SimpleGrantedAuthority;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.stereotype.Component;

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 java.io.IOException;
import java.util.ArrayList;
import java.util.List;

@Component
public class MyFilter implements Filter {

    @Autowired private JWTUtil jwtUtil;

    //=====
    // DO FILTER
    //=====

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

        //CAST REQUEST TO GET ACCESS TO HEADERS
        HttpServletRequest httpRequest = (HttpServletRequest) request;

        //GET AUTHORIZATION HEADER
        String authorizationHeader = httpRequest.getHeader("Authorization");

        //IF AUTHORIZATION HEADER PRESENT USE JWT TO PUT AUTHENTICATION OBJECT INTO CONTEXT
        if(authorizationHeader != null) { addAuthenticationObjectIntoContext(authorizationHeader); }

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

    //=====
    // ADD AUTHENTICATION OBJECT INTO CONTEXT
    //=====

    private void addAuthenticationObjectIntoContext(String authorizationHeader) {

        //EXTRACT JWT FROM AUTHORIZATION HEADER
        String jwt = jwtUtil.extractJWTFromAuthorizationHeader(authorizationHeader);

        //GET CLAIMS
        Claims claims = jwtUtil.getClaims(jwt);
        String username = (String) claims.get("username");
        String authoritiesJWT = (String) claims.get("authorities"); //"[book.read, book.delete]"

        //CREATE AUTHORITIES
        String authoritiesString = authoritiesJWT.replace("[", "").replace("]", "").replace(" ", "");
        String[] authoritiesArray = authoritiesString.split(",");
    }
}
```

```

        List<GrantedAuthority> authorities = new ArrayList<GrantedAuthority>();
        for(String authority : authoritiesArray) {
            authorities.add(new SimpleGrantedAuthority(authority));
        }

        //AUTHENTICATE
        Authentication authentication = new UsernamePasswordAuthenticationToken(username, null, authorities);

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

    }

}

```

WebSecurityConfig.java

```

package com.ivoronline.springboot_security_jwt.config;

import org.springframework.beans.factory.annotation.Autowired;
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.EnableWebSecurity;
import org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;
import org.springframework.security.config.http.SessionCreationPolicy;
import org.springframework.security.web.authentication.UsernamePasswordAuthenticationFilter;

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

    @Autowired private MyFilter myFilter;

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

        //ANONYMOUS ACCESS
        httpSecurity.authorizeRequests().antMatchers("/CreateJWT").permitAll(); //To get JWT
        httpSecurity.authorizeRequests().antMatchers("/GetClaims").permitAll(); //For Testing
        httpSecurity.authorizeRequests().antMatchers("/GetUsername").permitAll(); //For Testing

        //OTHER CONFIGURATION
        httpSecurity.csrf().disable(); //Enables POST
        httpSecurity.authorizeRequests().anyRequest().authenticated(); //Authenticated
        httpSecurity.addFilterBefore(myFilter, UsernamePasswordAuthenticationFilter.class); //Add Filter
        httpSecurity.sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS); //No Session

    }

}

```

Results

<http://localhost:8080/CreateJWT>



Postman

GET

<http://localhost:8080/CreateBook>

Headers

(add Key-Value)

Authorization: Bearer
eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJUZXN0SlDUUiwiwicm9sZSI6IiJPTEVfVVNFUiIsImlzcyI6Iml2b3JvbmxpbmUiLCJqdGkiOiIxIowi dXNlcml5hbWUiOjteXVzZXIifQ.g7CcreAHQvSR1JjQJqTBvT3eGcpz9kcuV0kgUFREiIc

Postman

The screenshot shows the Postman application window. At the top, there's a navigation bar with File, Edit, View, Help, Home, Workspaces, Reports, Explore, and a search bar. Below the navigation is a toolbar with various icons. The main area shows a list of collections, with 'JWT / http://localhost:8080/CreateBook' selected. Underneath, a specific request is defined: a GET method to 'http://localhost:8080/CreateBook'. The 'Headers' tab is active, showing the 'Authorization' header with the value 'Bearer eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJUZXN0SlDUUiwiwicm9sZSI6IiJPTEVfVVNFUiIsImlzcyI6Iml2b3JvbmxpbmUiLCJqdGkiOiIxIowi dXNlcml5hbWUiOjteXVzZXIifQ.g7CcreAHQvSR1JjQJqTBvT3eGcpz9kcuV0kgUFREiIc'. Other tabs like Params, Body, and Cookies are visible but not selected. At the bottom, the response status is shown as 200 OK with a size of 372 B.

1.11.6 Step 6 - Authenticate

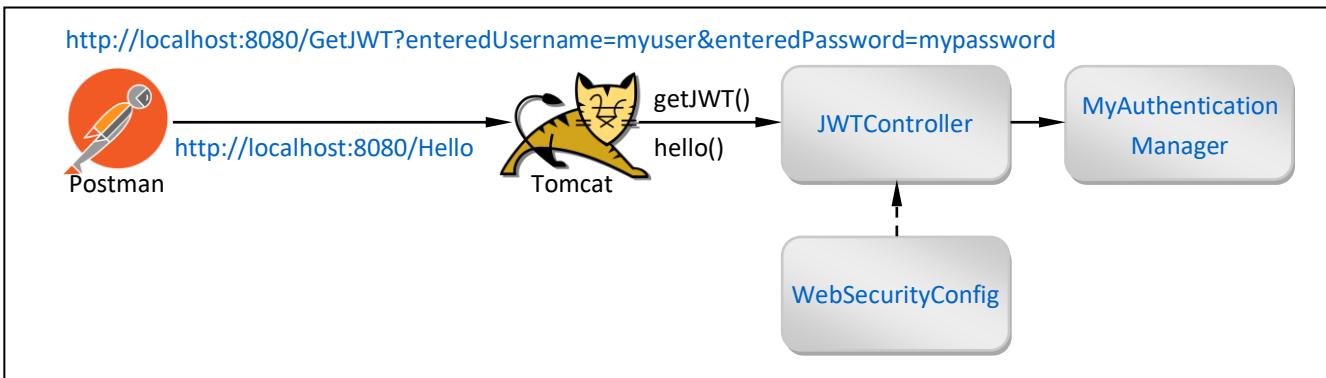
Info

[G]

- Up until this point we have been using [/CreateJWT](#) to return JWT with hard coded data and without any Authentication. In other words we didn't provide any Credentials and we didn't compare them with stored Credentials to identify User.
- So in this last step we will add this missing part and we will return JWT only if User is Authenticated. For this purpose we will create new [/Authenticate](#) Endpoint that will compare provided Credentials with stored ones.

Application Schema

[Results]



Procedure

- Inside Package: config
- Create Class: [MyAuthenticationManager.java](#) (Compare entered Credentials with stored ones)
- Edit Class: [JWTController.java](#) (Add anonymous Endpoint "/Authenticate")
- Edit Class: [WebSecurityConfig.java](#) (Allow access to anonymous Endpoint "/Authenticate")

MyAuthenticationManager.java

```
package com.ivoronline.springboot_security_jwt.config;

import org.springframework.security.authentication.AuthenticationManager;
import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;
import org.springframework.security.core.Authentication;
import org.springframework.security.core.GrantedAuthority;
import org.springframework.security.core.authority.SimpleGrantedAuthority;
import org.springframework.stereotype.Component;
import java.util.ArrayList;
import java.util.List;

@Component
public class MyAuthenticationManager implements AuthenticationManager {

    @Override
    public Authentication authenticate(Authentication enteredAuthentication) {

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

        //GET STORED CREDENTIALS
        //Here they are hard coded (for simplicity reason)
        //Call UserDetailsService(enteredUsername) to get UserDetails with Password & Authorities from DB
        String storedUsername      = "admin";
        String storedPassword      = "adminpassword";
        String storedAuthorities   = "book.create, book.delete";

        //AUTHENTICATE USER (COMPARE ENTERED AND STORED CREDENTIALS)
        if (!enteredUsername.equals(storedUsername)) { System.out.println("Username not found"); return null; }
        if (!enteredPassword.equals(storedPassword)) { System.out.println("Incorrect Password"); return null; }

        //CREATE AUTHORITIES
        String[] authoritiesArray = storedAuthorities.split(", ");
        List<GrantedAuthority> authorities = new ArrayList<GrantedAuthority>();
        for(String authority : authoritiesArray) {
            authorities.add(new SimpleGrantedAuthority(authority));
        }

        //CREATE VALIDATED AUTHENTICATION
        Authentication validatedAuth = new UsernamePasswordAuthenticationToken(enteredUsername, null,
        authorities);

        //RETURN VALIDATES AUTHENTICATION
        return validatedAuth;
    }
}
```

```

package com.ivorononline.springboot_security_jwt.controllers;

import com.ivorononline.springboot_security_jwt.config.JWTUtil;
import com.ivorononline.springboot_security_jwt.config.MyAuthenticationManager;
import io.jsonwebtoken.Claims;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;
import org.springframework.security.core.Authentication;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestHeader;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.ResponseBody;

@Controller
public class JWTController {

    @Autowired
    JWTUtil jwtUtil;

    @Autowired
    MyAuthenticationManager myAuthenticationManager;

    //=====
    // GET JWT
    //=====

    @ResponseBody
    @RequestMapping("/GetJWT")
    public String getJWT(@RequestParam String enteredUsername, @RequestParam String enteredPassword) {

        //AUTHENTICATE (COMPARE ENTERED AND STORED CREDENTIALS)
        Authentication enteredAuth = new UsernamePasswordAuthenticationToken(enteredUsername, enteredPassword);
        Authentication returnedAuth = myAuthenticationManager.authenticate(enteredAuth);

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

        //CREATE JWT
        String username = (String) returnedAuth.getPrincipal();
        String role = (String) returnedAuth.getAuthorities().toString().replace("[", "").replace("]", "").replace(",","");
        String jwt = jwtUtil.createJWT(username, role);

        //RETURN JWT
        return jwt;
    }

    //=====
    // CREATE JWT
    //=====

    @ResponseBody
    @RequestMapping("/CreateJWT")
    public String createJWT() {

        //CREATE JWT
        String jwt = jwtUtil.createJWT("admin", "[book.create, book.delete]");

        //RETURN JWT
        return jwt;
    }

    //=====
    // GET CLAIMS
    //=====
}

```

```
@ResponseBody
@RequestMapping("/GetClaims")
public Claims getClaims(@RequestHeader("Authorization") String authorization) {

    //EXTRACT JWT FROM AUTHORIZATION HEADER
    String jwt = jwtUtil.extractJWTFromAuthorizationHeader(authorization);

    //GET CLAIMS
    Claims claims = jwtUtil.getClaims(jwt);

    //RETURN CLAIMS
    return claims;

}

//=====
// GET USERNAME
//=====
@ResponseBody
@RequestMapping("/GetUsername")
public String getUsername(@RequestHeader("Authorization") String authorization) {

    //EXTRACT JWT FROM AUTHORIZATION HEADER
    String jwt = jwtUtil.extractJWTFromAuthorizationHeader(authorization);

    //GET USERNAME
    String username = jwtUtil.getUsername(jwt);

    //RETURN USERNAME
    return username;

}
}
```

WebSecurityConfig.java

```
package com.ivoronline.springboot_security_jwt.config;

import org.springframework.beans.factory.annotation.Autowired;
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.EnableWebSecurity;
import org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;
import org.springframework.security.config.http.SessionCreationPolicy;
import org.springframework.security.web.authentication.UsernamePasswordAuthenticationFilter;

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

    @Autowired private MyFilter myFilter;

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

        //ANONYMOUS ACCESS
        httpSecurity.authorizeRequests().antMatchers("/GetJWT").permitAll(); //To get Authenticated JWT
        httpSecurity.authorizeRequests().antMatchers("/CreateJWT").permitAll(); //To get hard coded JWT
        httpSecurity.authorizeRequests().antMatchers("/GetClaims").permitAll(); //For Testing
        httpSecurity.authorizeRequests().antMatchers("/GetUsername").permitAll(); //For Testing

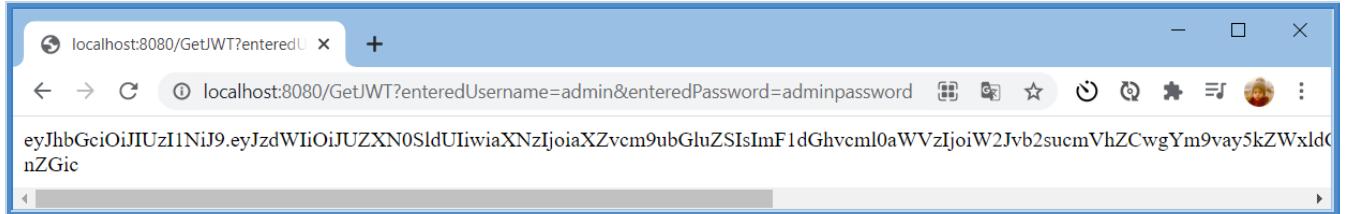
        //OTHER CONFIGURATION
        httpSecurity.csrf().disable(); //Enables POST
        httpSecurity.authorizeRequests().anyRequest().authenticated(); //Authenticated
        httpSecurity.addFilterBefore(myFilter, UsernamePasswordAuthenticationFilter.class); //Add Filter
        httpSecurity.sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS); //No Session

    }

}
```

Results

<http://localhost:8080/GetJWT?enteredUsername=admin&enteredPassword=adminpassword>



Postman

GET

<http://localhost:8080/CreateBook>

Headers

(add Key-Value)

Authorization: Bearer
eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJUZXN0SldUIiwicm9sZSI6IiJPTEVfVVNFUiIsImlzcyI6Iml2b3JvbmxpbmUiLCJqdGkiOiIxIiwi dXNlcm5hbWUiOiJteXvzZXIfQ.g7CcreAHQvSR1JjQJqTBvT3eGcpz9kcuV0kgUFREiIc

Postman

The screenshot shows the Postman application interface. A GET request is being made to <http://localhost:8080/CreateBook>. In the Headers section, the Authorization header is set to Bearer followed by a long JWT token. The response status is 200 OK.

2 Demo Applications

Info

- Following tutorials contain demo applications that show how to combine functionalities covered in previous tutorials.
- Each application is broken into multiple tutorials/steps showing how to add additional functionalities.

Balance between simplicity and coding practices

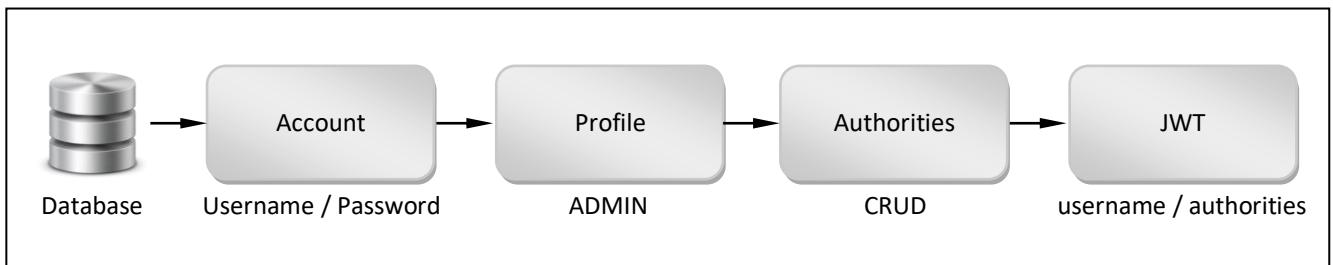
- In the previous tutorials main priority was simplicity, sometimes at the expense of the best coding practices.
For instance in previous tutorials Entities were created by using **public Properties** to keep them as simple as possible by avoiding having setters and getters.
- Tutorials that follow will try to strike a balance between remaining simple while following the best coding practices.
In case of Entities this means that we will have **private Properties** accessed through Lombok's setters and getters.
This Results in a more complex code and more stuff to learn (like Lombok) but it is more aligned with real life practices.
But it will be far easier to make this transition after you have already learned the basics in previous tutorials.

2.1 JWT Authorities from DB

Info

- This tutorial shows how to combine following tutorials
 - [Authorities - DB](#) Account is given Authorities through Profile assigned to that Account
 - [JWT \(JSON Web Token\)](#) will be modified so that Authorities get loaded into JWT (instead of hard coded Role)
- JWT contains Authorities that are related to the Profile assigned to the Account (Authorities are taken from DB)
JWT Authorities are later used to create Authenticated Object and store it into Context (DB is not used anymore)
- If Request doesn't contain JWT you will be able to authenticate through Standard Login Form.
If you want to use JWT you can first Authenticate by calling [/GetJWT](#) with Username and Password:
<http://localhost:8080/GetJWT?enteredUsername=admin&enteredPassword=adminpassword>
After that you can use JWT inside [Authentication Header](#) to access Endpoints without logging in again.
- We will start by copying [Authorities - DB](#) tutorial into Package `security.authoritiesfromdb`.
We will make no changes to this code. It will allow us to authenticate through Standard Login Form without JWT.
Then we will copy [JWT \(JSON Web Token\)](#) tutorial into Package `security.jwt`.
After that we just need to modify [JWT \(JSON Web Token\)](#) to take Authorities from DB (instead of using hard coded Role).

Authorities are loaded into JWT



JWT

```
{  
    "username" : "admin",  
    "authorities" : "[book.create, book.read, book.update, book.delete]"  
}
```

MyController.java

```
@Controller  
public class MyController {  
  
    @ResponseBody  
    @RequestMapping("/CreateBook")  
    @PreAuthorize("hasAuthority('book.create')")  
    public String createBook() {  
        return "Only ADMIN can create Book";  
    }  
}
```

2.1.1 Step 1 - DB Authorities

Info

- In this step we will copy [Authorities - DB tutorial](#) into Package `security.authoritiesfromdb`.
- It will allow us to Authenticate through Standard Login Form without JWT.
- We will move the code under new `security.authoritiesfromdb` Package.
- But we will make no changes to the code itself.

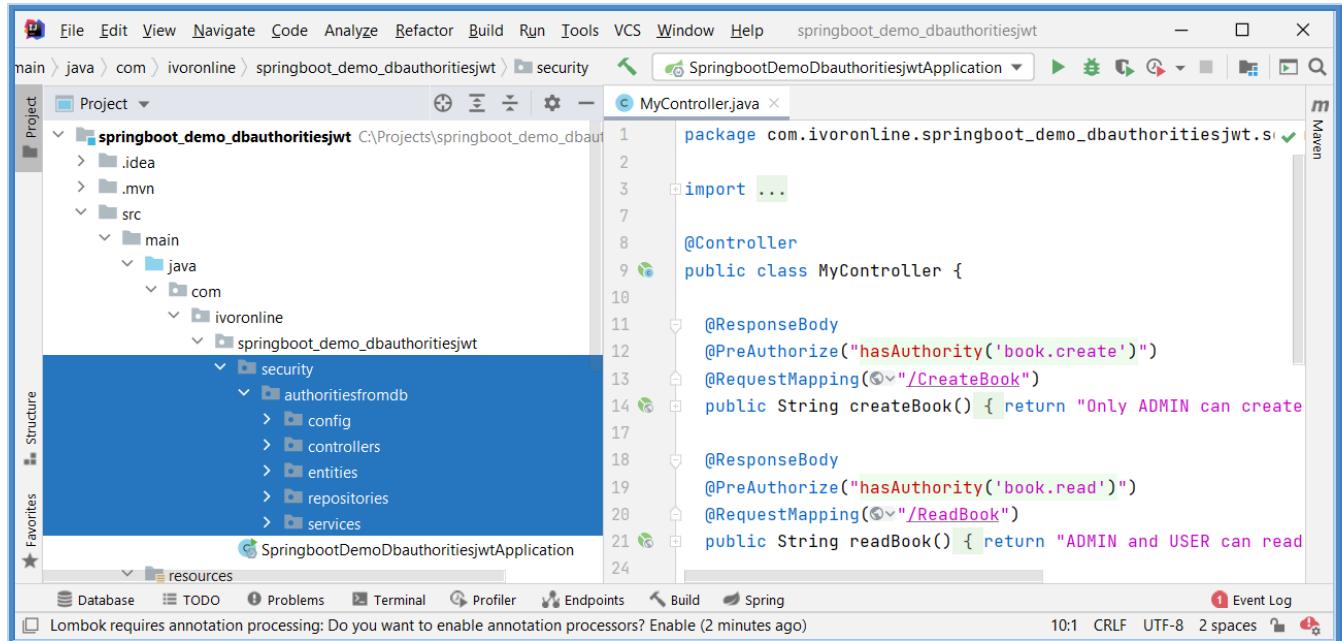
Spring Boot Starters

GROUP	DEPENDENCY	DESCRIPTION
Web	Spring Web	Enables <code>@Controller</code> , <code>@RequestMapping</code> and Tomcat Server
Security	Spring Security	Enables Spring Security
SQL	Spring Data JPA	Enables <code>@Entity</code> and <code>@Id</code>
SQL	H2 Database	Enables in-memory H2 Database

Procedure

- Create Project: `springboot_demo_jwtauthoritiesfromdb` (add Spring Boot Starters from the table)
- Create Package: **security** (inside main package)
- Create Package: **authoritiesfromdb** (inside security package)
- Move existing Packages into `authoritiesfromdb` Package
- Reopen Project to fix [imports](#)

Application Structure



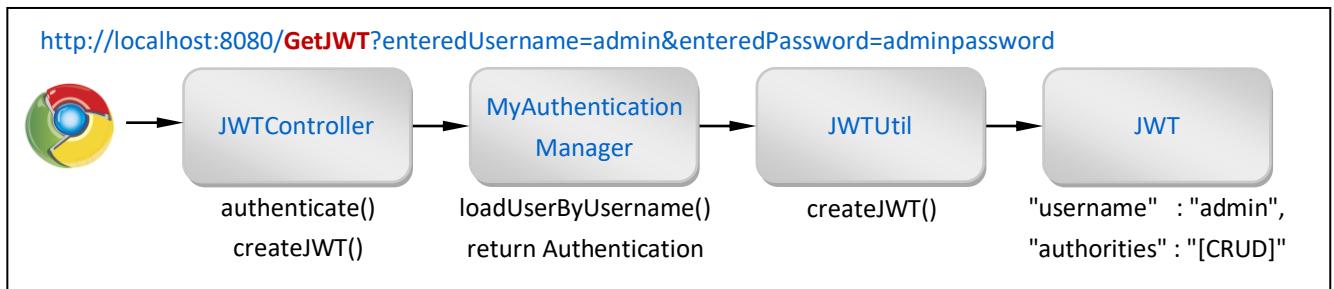
2.1.2 Step 2 - JWT

Info

- In this step we will
 - copy [JWT \(JSON Web Token\) tutorial](#) into Package `security.jwt`
 - modify following Classes to take Authorities from DB (instead of using hard coded Role)

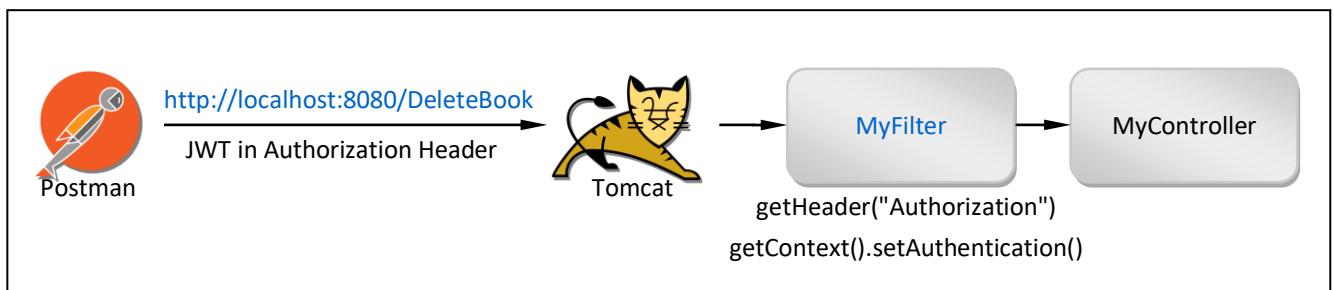
Get JWT

[[Results](#)]



Send JWT

([Filter](#) gets Authorities from JWT & stores Authentication Object into Context)



Procedure

- Edit File: pom.xml (add dependencies)
- Create Package: jwt (inside Package security)
- Create Class: JWTUtil.java (Compare entered Credentials with stored ones)
- Create Class: MyAuthenticationManager.java (Compare entered Credentials with stored ones)
- Create Class: JWTController.java (Compare entered Credentials with stored ones)
- Create Class: MyFilter.java (Add anonymous Endpoint "/Authenticate")

pom.xml

```
<dependency>
    <groupId>io.jsonwebtoken</groupId>
    <artifactId>jjwt</artifactId>
    <version>0.9.1</version>
</dependency>

<dependency>
    <groupId>javax.xml.bind</groupId>
    <artifactId>jaxb-api</artifactId>
    <version>2.3.0</version>
</dependency>
```

JWTUtil.java

```
package com.ivorononline.springboot_demo_jwtauthoritiesfromdb.security.jwt;

import io.jsonwebtoken.Claims;
import io.jsonwebtoken.JwtBuilder;
import io.jsonwebtoken.Jwts;
import io.jsonwebtoken.SignatureAlgorithm;
import org.springframework.stereotype.Component;

import javax.crypto.spec.SecretKeySpec;
import javax.xml.bind.DatatypeConverter;
import java.security.Key;
import java.util.HashMap;
import java.util.Map;

@Component
public class JWTUtil {

    //USED TO CREATE & DECODE JWT
    public final static String SECRET_KEY = "mysecretkey";

    //=====
    // CREATE JWT
    //=====

    public String createJWT(String username, String role) {

        //HEADER (SPECIFY ALGORITHM)
        SignatureAlgorithm signatureAlgorithm = SignatureAlgorithm.HS256;

        //PAYLOAD (SPECIFY CLAIMS)
        Map<String, Object> customClaims = new HashMap<>();
        customClaims.put("username", username);
        customClaims.put("authorities", role);

        JwtBuilder builder = Jwts.builder().setClaims(customClaims);

        //SIGNATURE (SPECIFY SECRET KEY)
        byte[] apiKeySecretBytes = DatatypeConverter.parseBase64Binary(SECRET_KEY);
        Key signingKey = new SecretKeySpec(apiKeySecretBytes, signatureAlgorithm.getJcaName());
    }
}
```

```

//EXTRACT JWT
String jwt = builder.signWith(signatureAlgorithm, signingKey).compact();

//RETURN JWT
return jwt;

}

//=====
// EXTRACT JWT FROM AUTHORIZATION HEADER
//=====

public String extractJWTFromAuthorizationHeader(String authorization) {

    //GET AUTHORIZATION HEADER
    if (authorization == null || !authorization.startsWith("Bearer ")) {
        System.out.println("Authorization Header not found");
        return null;
    }

    //EXTRACT JWT
    String jwt = authorization.substring(7);

    //RETURN JWT
    return jwt;

}

//=====
// GET CLAIMS
//=====

public Claims getClaims(String jwt) {

    //GET CLAIMS
    Claims claims = Jwts.parser()
        .setSigningKey(DatatypeConverter
            .parseBase64Binary(SECRET_KEY))
        .parseClaimsJws(jwt)
        .getBody();

    //RETURN CLAIMS
    return claims;

}

//=====
// GET USERNAME
//=====

public String getUsername(String jwt) {
    Claims claims = getClaims(jwt);
    String username = (String) claims.get("username");
    return username;
}
}

```

MyAuthenticationManager.java

```
package com.ivoronline.springboot_demo_jwtauthoritiesfromdb.security.jwt;

import com.ivoronline.springboot_demo_jwtauthoritiesfromdb.security.authoritiesfromdb.services.MyUserDetailsService;
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.stereotype.Component;

@Component
public class MyAuthenticationManager implements AuthenticationManager {

    @Autowired MyUserDetailsService myUserDetailsService;

    @Override
    public Authentication authenticate(Authentication enteredAuthentication) {

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

        //GET USER FROM DB
        UserDetails userDetails = myUserDetailsService.loadUserByUsername(enteredUsername);

        //AUTHENTICATE USER
        if (userDetails == null) { System.out.println("Username not found");}
        return null;
        if (!enteredPassword.equals(userDetails.getPassword())) { System.out.println("Incorrect Password");}
        return null;

        //CREATE VALIDATED AUTHENTICATION
        Authentication validatedAuth = new UsernamePasswordAuthenticationToken(enteredUsername, null,
        userDetails.getAuthorities());
        System.out.println("validatedAuth = " + validatedAuth);

        //RETURN VALIDATED AUTHENTICATION
        return validatedAuth;
    }
}
```

JWTController.java

```
package com.ivoronline.springboot_demo_jwtauthoritiesfromdb.security.jwt;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;
import org.springframework.security.core.Authentication;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.ResponseBody;

@Controller
public class JWTController {

    @Autowired JWTUtil jwtUtil;
    @Autowired MyAuthenticationManager myAuthenticationManager;

    //=====
    // GET JWT
    //=====

    @ResponseBody
    @RequestMapping("/GetJWT")
    public String getJWT(@RequestParam String enteredUsername, @RequestParam String enteredPassword) {

        //LOG
        System.out.println("GetJWT ===== ");

        //AUTHENTICATE
        Authentication enteredAuth = new UsernamePasswordAuthenticationToken(enteredUsername, enteredPassword);
        Authentication returnedAuth = myAuthenticationManager.authenticate(enteredAuth);

        System.out.println("returnedAuth = " + returnedAuth);

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

        //CREATE JWT
        String username = (String) returnedAuth.getPrincipal();
        String authorities = returnedAuth.getAuthorities().toString();

        System.out.println("authorities = " + authorities);

        String jwt = jwtUtil.createJWT(username, authorities);

        //RETURN JWT
        return jwt;
    }
}
```

MyFilter.java

```
package com.ivorononline.springboot_demo_jwtauthoritiesfromdb.security.jwt;

import io.jsonwebtoken.Claims;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;
import org.springframework.security.core.Authentication;
import org.springframework.security.core.GrantedAuthority;
import org.springframework.security.core.authority.SimpleGrantedAuthority;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.stereotype.Component;

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 java.io.IOException;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;

@Component
public class MyFilter implements Filter {

    @Autowired private JWTUtil jwtUtil;

    //=====
    // DO FILTER
    //=====

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

        //LOG
        System.out.println("doFilter =====");

        //CAST REQUEST TO GET ACCESS TO HEADERS
        HttpServletRequest httpRequest = (HttpServletRequest) request;

        //GET AUTHORIZATION HEADER
        String authorizationHeader = httpRequest.getHeader("Authorization");

        //IF AUTHORIZATION HEADER EXISTS USE JWT TO PUT AUTHENTICATION OBJECT INTO CONTEXT
        if(authorizationHeader != null) { addAuthenticationObjectIntoContext(authorizationHeader); }

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

    //=====
    // ADD AUTHENTICATION OBJECT INTO CONTEXT
    //=====

    private void addAuthenticationObjectIntoContext(String authorizationHeader) {

        //EXTRACT JWT FROM AUTHORIZATION HEADER
        String jwt = jwtUtil.extractJWTFromAuthorizationHeader(authorizationHeader);

        //GET CLAIMS
        Claims claims = jwtUtil.getClaims(jwt);
        String username = (String) claims.get("username");
        String authoritiesJWT = (String) claims.get("authorities"); //"[book.read, book.delete]"
    }
}
```

```
//CREATE AUTHORITIES
String authoritiesString = authoritiesJWT.replace("[","").replace("]","");
//book.read,book.delete"
String[] authoritiesArray = authoritiesString.split(",");
List<GrantedAuthority> authorities = new ArrayList<GrantedAuthority>();
for(String authority : authoritiesArray) {
    authorities.add(new SimpleGrantedAuthority(authority));
}

//AUTHENTICATE
Authentication authentication = new UsernamePasswordAuthenticationToken(username, null, authorities);

System.out.println("authentication = " + authentication);

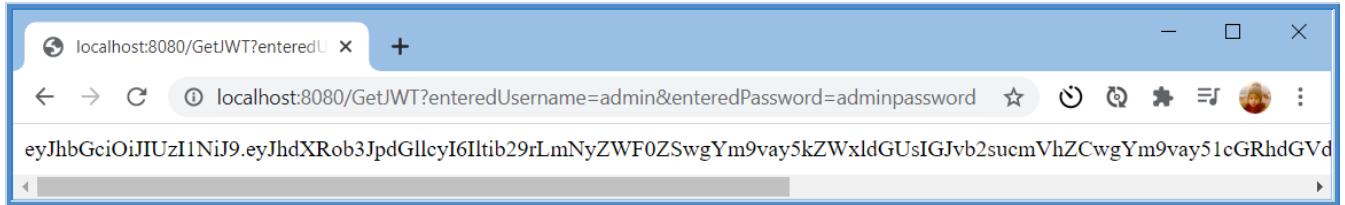
//STORE AUTHENTICATION INTO CONTEXT (SESSION)
SecurityContextHolder.getContext().setAuthentication(authentication);

}

}
```

Results

<http://localhost:8080/GetJWT?enteredUsername=admin&enteredPassword=adminpassword>



<https://jwt.io/#debugger>

(Decoded JWT)

```
{  
    "username" : "admin"  
    "authorities" : "[book.read, book.delete]",  
    "iss" : "ivoronline",  
    "sub" : "TestJWT",  
    "jti" : "1"  
}
```

Postman - JWT in Authorization Header

GET or POST

<http://localhost:8080/ReadBook>

Headers

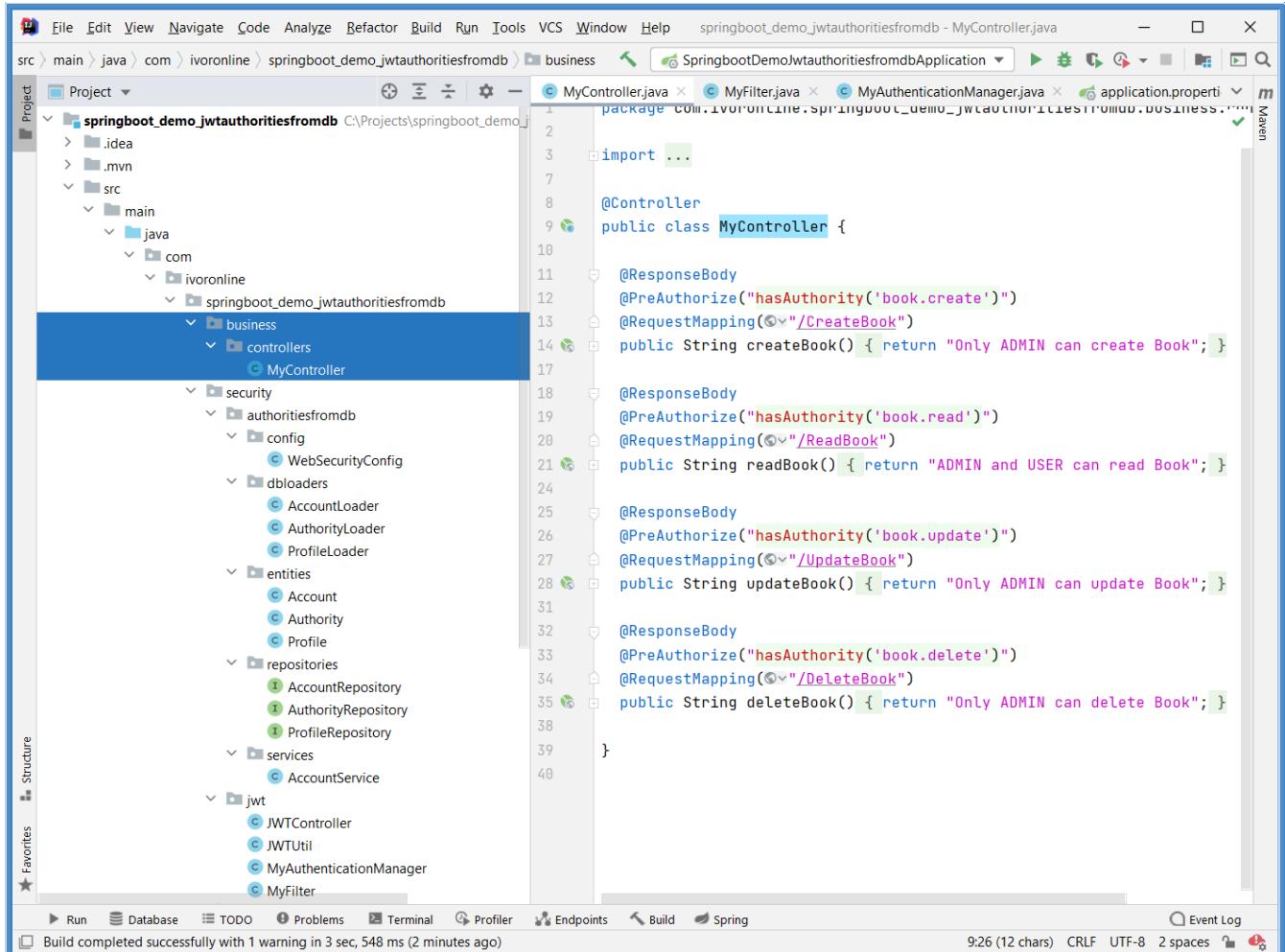
(add Key-Value)

```
Authorization: Bearer  
eyJhbGciOiJIUzI1NiJ9.eyJdWIIoIjUZXN0SldUiwiwcm9sZSI6IiJPTEVfVVNFUiIsImlzcyI6Iml2b3JvbmxpbmUiLCJqdGkiOiIxIiwidXNlcm5hbWUiOiJteXVzZXIifQ.g7CcreAHQvSR1JjQJqTBvT3eGcpz9kcuV0kgUFREiIc
```

Postman

A screenshot of the Postman application. The top navigation bar shows 'Postman' and various menu options like File, Edit, View, Help. The main interface shows a list of requests under 'JWT / http://localhost:8080/ReadBook'. A 'GET' request is selected, pointing to 'http://localhost:8080/ReadBook'. In the 'Headers' tab, there is an 'Authorization' header set to 'Bearer eyJhbGciOiJIUzI1NiJ9.eyJdWIIoIjUZXN0SldUiwiwcm9sZSI6IiJPTEVfVVNFUiIsImlzcyI6Iml2b3JvbmxpbmUiLCJqdGkiOiIxIiwidXNlcm5hbWUiOiJteXVzZXIifQ.g7CcreAHQvSR1JjQJqTBvT3eGcpz9kcuV0kgUFREiIc'. The response status is '200 OK' with a time of '38 ms' and a size of '366 B'. The response body shows the message 'ADMIN can read books'.

Application Structure



pom.xml

```
<dependencies>

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

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

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

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

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

```
<dependency>
    <groupId>io.jsonwebtoken</groupId>
    <artifactId>jjwt</artifactId>
    <version>0.9.1</version>
</dependency>

<dependency>
    <groupId>javax.xml.bind</groupId>
    <artifactId>jaxb-api</artifactId>
    <version>2.3.0</version>
</dependency>

</dependencies>
```

3 Appendix

Info

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

3.1 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.

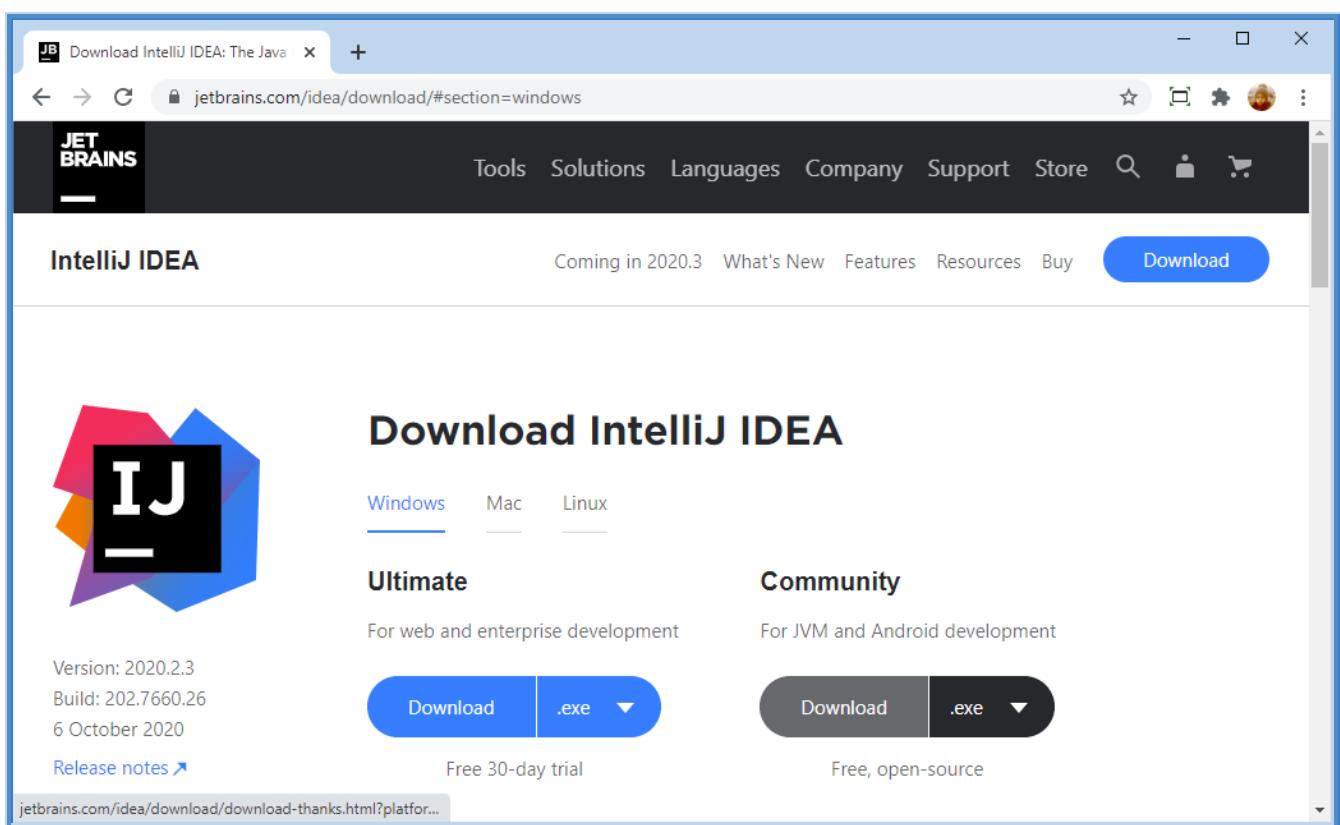
3.1.1 Install

Info

- This tutorial shows how to install IntelliJ IDEA 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

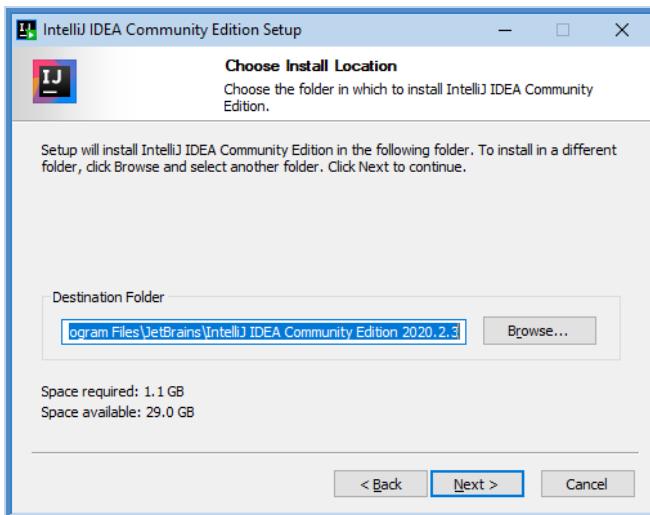
Run



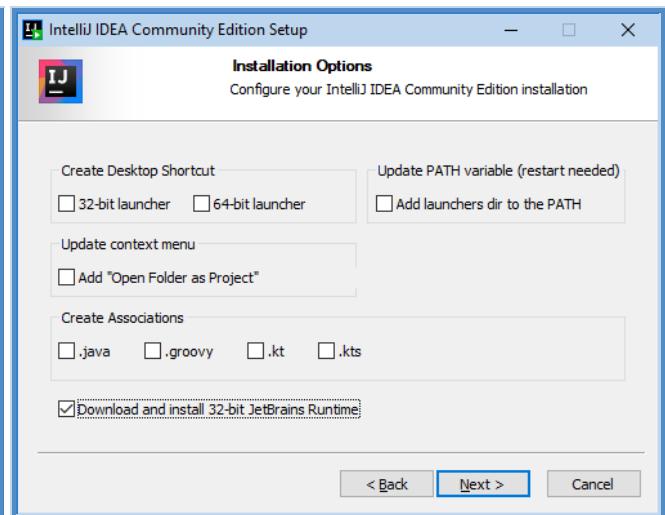
Next



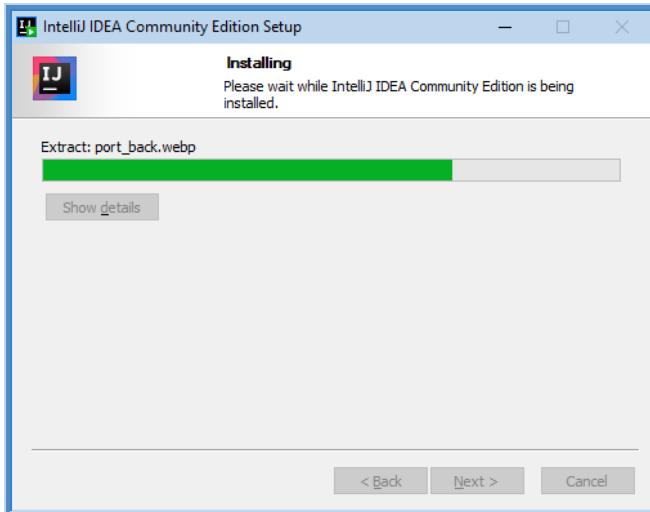
Next



Next



Progress



Finish



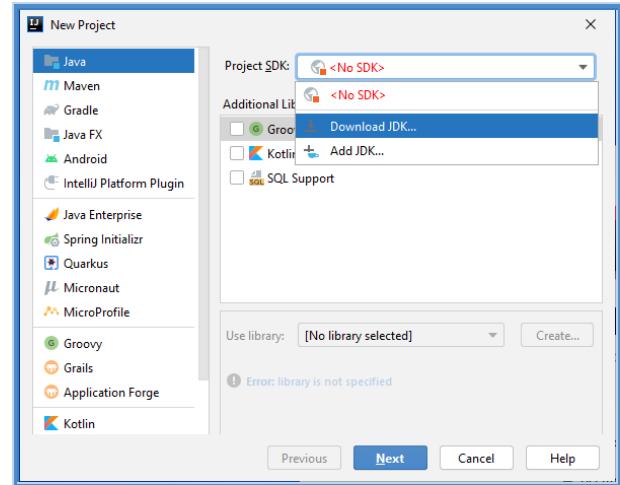
Download Java JDK

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

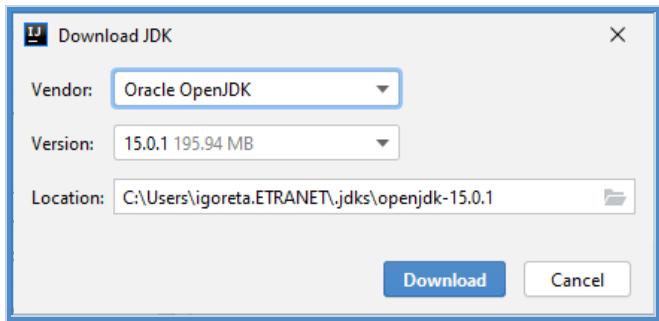
New Project



Download JDK



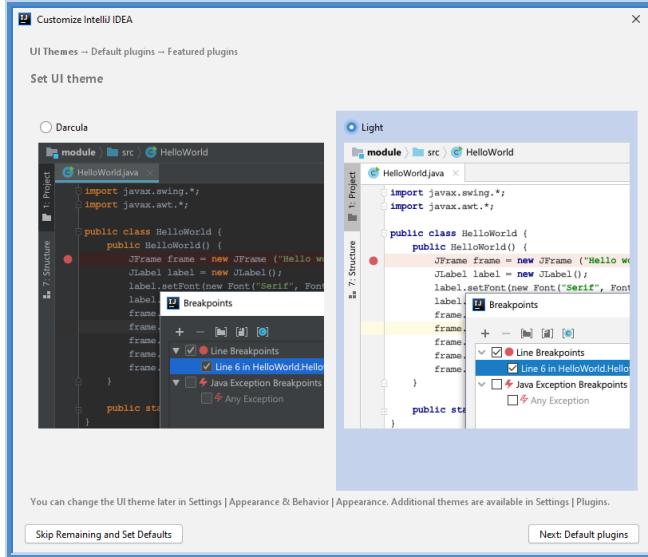
Download



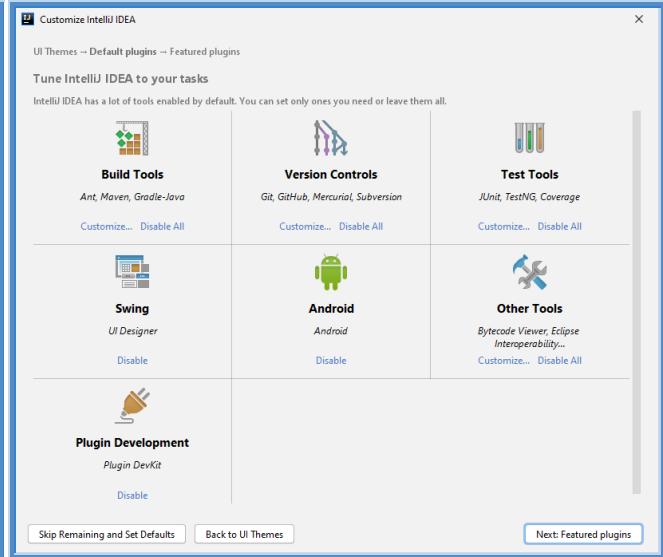
Customize

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

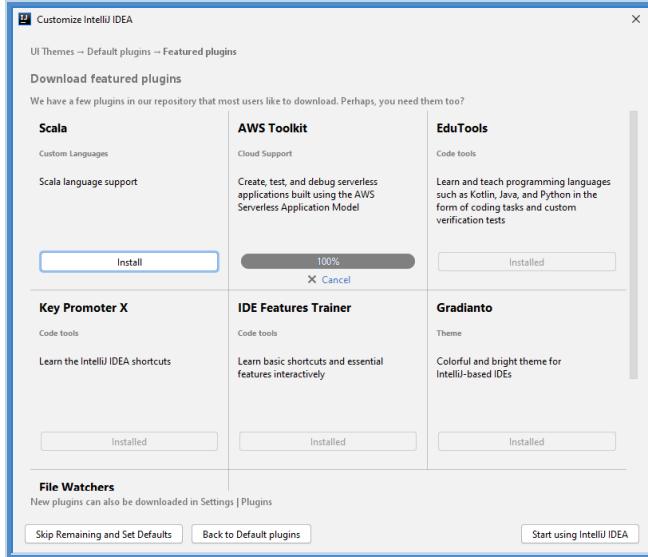
Light



Next



Start



3.1.2 Create Project

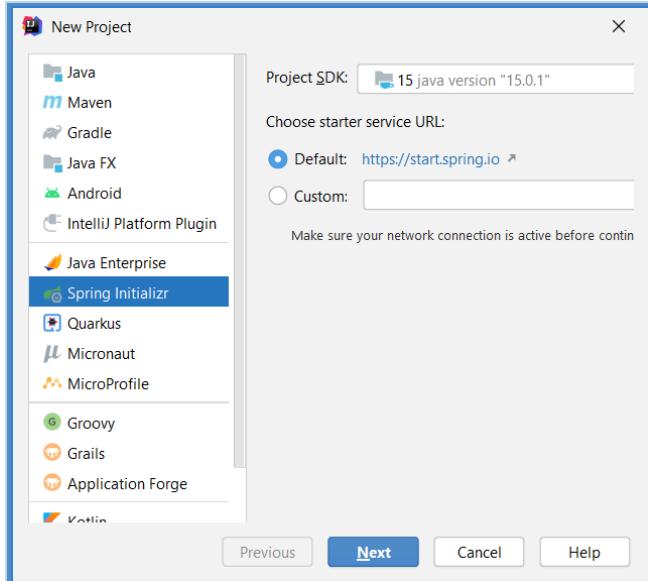
Info

- 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.](#)

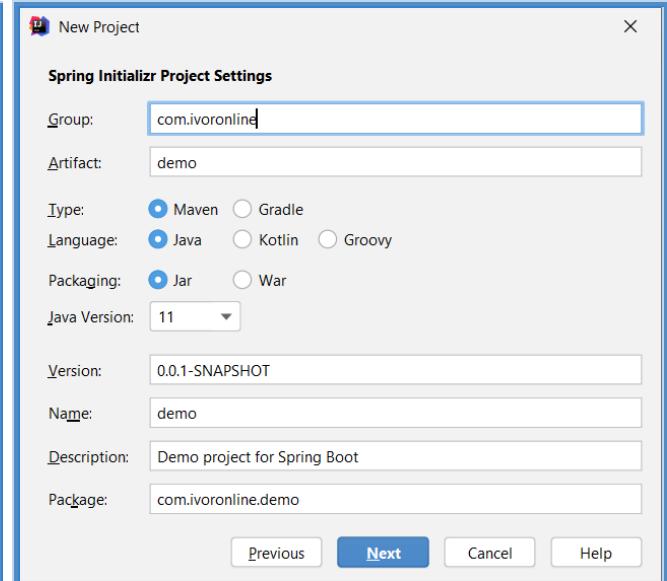
Create Spring Boot Project

- 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)

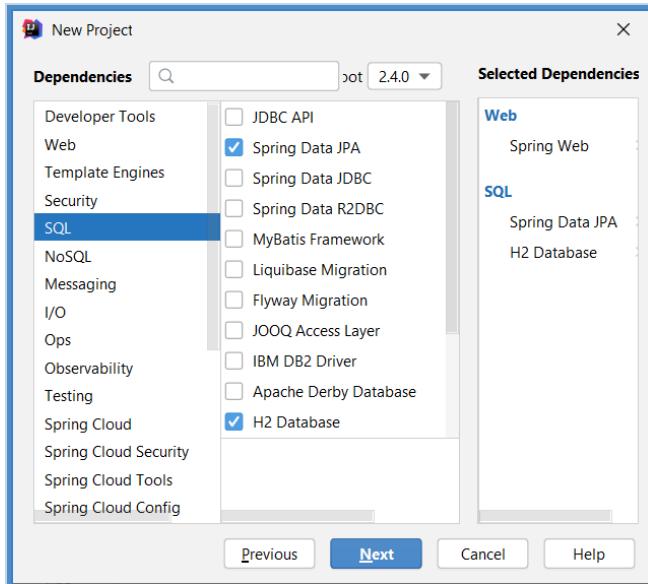
File - New - Project - Spring Initializr



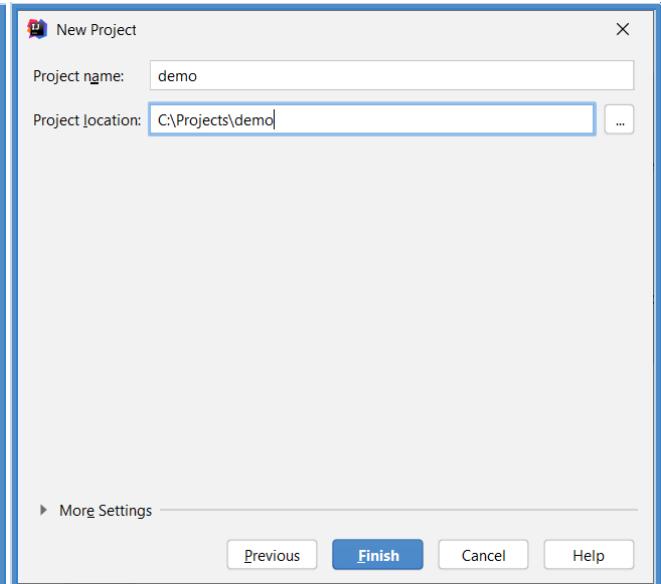
Project Settings



Dependencies



Project name



3.1.3 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;

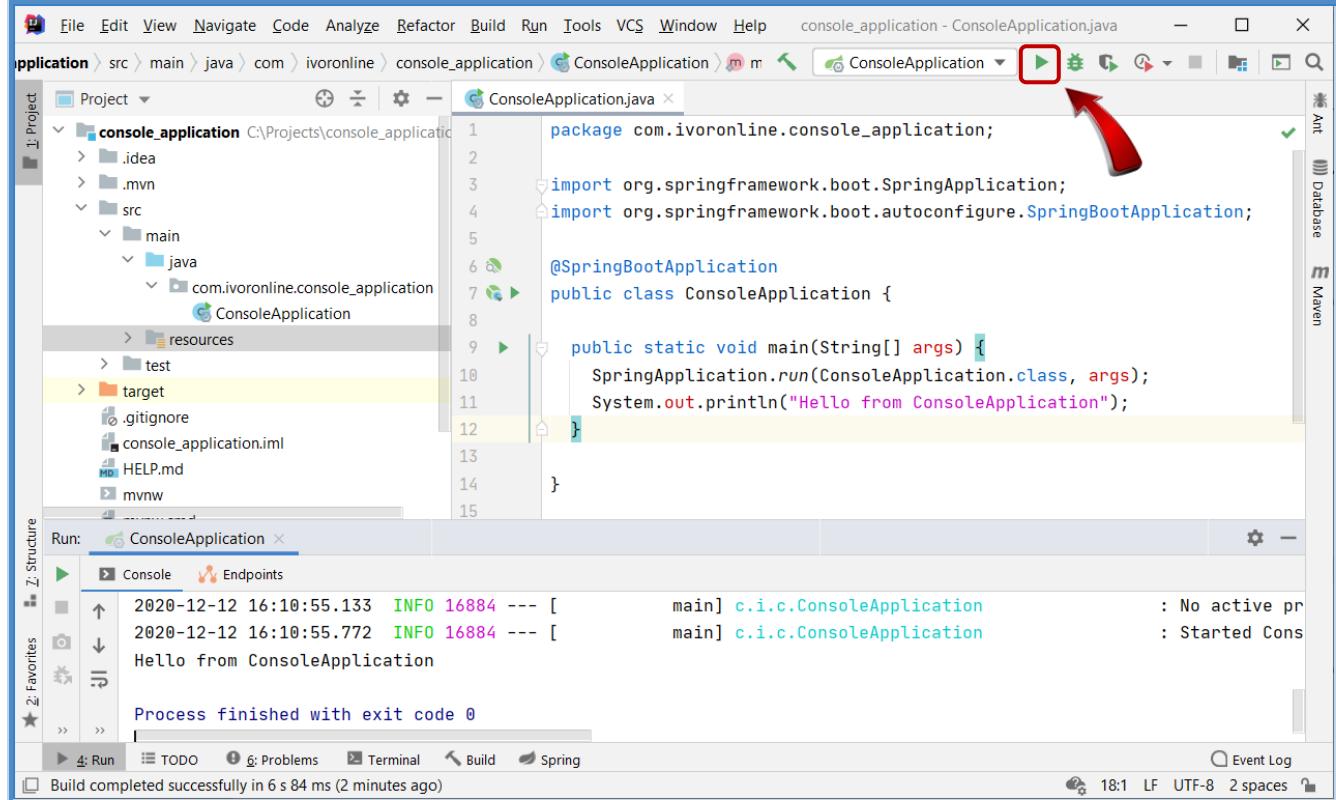
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@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
```

3.2 JWT

Info

- Following tutorials show how to use **JSON Web Token** (JWT).
- <https://jwt.io> is Web Page that allows you to Encode and Decode JWT.
- Decoded JWT consist of three parts (shown on the right)
 - Header & Payload** are JSON Objects
 - Signature** is String created from: Header, Payload and Secret
- Each part is then Base64Url encoded and connected with dots '.' to create resulting Encoded JWT (shown on the left).

Encoded JWT

```
Header.Payload.Signature
```

<https://jwt.io/#debugger>

The screenshot shows the jwt.io debugger interface. At the top, the URL is https://jwt.io/#debugger. Below the header, there's a dropdown menu for the algorithm set to HS512. The interface is divided into two main sections: 'Encoded' on the left and 'Decoded' on the right.

Encoded: This section displays the raw encoded JWT string:
eyJhbGciOiJIUzUxMiIsInR5cCI
6IkpXVCJ9.eyJleHAiOjE1NjE4N
TI4MDAsImRvbWFpbIi6InNvbWUt
ZG9tYWluIiwidGxjSWRlbnRpZml
lcIi6InNvbWUtdGxjIn0.AvhAYU
MX4t7_imMd8IoSvTFRzTcAUuc5b
Ak9G5_kt1Eq2fPA6Faa5LLrYJlz
Z2C7XK8du_FCSzx8-wh3n0Lqaw

Decoded: This section shows the decoded components:
HEADER:
{
 "alg" : "HS512",
 "typ" : "JWT"
}

PAYOUT:
{"exp" : 1561852800,
 "domain" : "some-domain",
 "tlcIdentifier" : "some-tlc"}

VERIFY SIGNATURE
HMACSHA512(
 base64UrlEncode(header) + "." +
 base64UrlEncode(payload),
 GtRpJ0Gtmh29GiCcQWrv
) secret base64 encoded

3.2.1 Introduction

Info

[R]

- ➊ JWT
 - stand for **JSON Web Token**
 - is used to **securely** transmit information between parties (as JSON Object whose content can't be tampered with)
(like Authorities/Roles/URLs User can access)
- ➋ The main purpose of JWT is to control **Authorization** (access to Restricted Resources).
This means that Authorities/Roles should be defined as Claims inside JWT itself.
When JWT is part of Request there is no need to go into DB to Authenticate User and get its Authorities.
- ➌ JWT doesn't have to contain Username unless
 - Username is needed for logging purposes
 - Authorization depends on Username - like in multitenancy applications when User can only read its own Emails/Books
- ➍ JWT is used as a single sign on solution
 - <http://localhost:8080/Authenticate> is used to Authenticate User and return JWT with Authorities/Roles as Claims
 - <http://localhost:8080>Hello> Restricted Resource can then be accessed by just providing JWT with Authorities
- ➎ JWT is designed so that **User can't change Authorities** that are inside JWT.
This is achieved by using secret key which is needed to properly Encode/Decode JWT which is stored inside Application.
But **JWT can be stolen** and use it to access restricted resource it doesn't have access to.
For that reason JWT Tokens usually have short expiration period of less than a day.
- ➏ JWT can be **digitally signed** using
 - secret (with the HMAC algorithm)
 - public/private key pair (using RSA or ECDSA)
- ➐ JWT can be
 - **Encrypted** which hides claims from other parties
 - **Non-Encrypted** when information is exposed to other parties (but they are unable to change it)

Purpose

- ➊ JWT is used for Authorization and Information Exchange.

Authorization

- ➋ JWT is mostly used for Single Sign On Authorization because of its
 - small overhead
 - ability to be easily used across different domains.
- ➌ Once user logs in, each subsequent request will include JWT allowing user to access resources permitted with that token.

Information Exchange

- ➍ JSON Web Tokens are a good way of securely transmitting information between parties.
- ➎ Because JWTs can be signed you are sure senders are who they say they are (for example using public/private key pairs)
- ➏ You can also verify that content hasn't been tampered with (since the signature is calculated using header and payload)

3.2.2 Structure

Info

- JWT structure consist of three parts separated by dots
 - Header
 - Payload
 - Signature

JWT Structure

```
Header.Payload.Signature
```

Header

- Header is first part of JWT and consists of two parts
 - type of the token (which is JWT)
 - signing algorithm being used (like HMAC SHA256 or RSA)
- Header JSON is Base64Url encoded to form the first part of JWT.

Header example

```
{  
  "alg": "HS256",  
  "typ": "JWT"  
}
```

Payload

- Payload is second part of JWT and contains Claims.
- Claims are statements about the User and additional data.
- There are three types of claims
 - registered (predefined) sub (subject), exp (expiration time), jti (JWT ID)
 - public
 - private (custom) name (username), role
- Payload JSON is Base64Url encoded to form the second part of JWT.

Payload example

```
{  
  "sub" : "1234567890",  
  "name" : "John Doe",  
  "admin" : true  
}
```

Signature

- Signature
 - is used to verify the message wasn't changed along the way
 - and, in the case of tokens signed with a private key, it can also verify that the sender of the JWT is who it says it is.
- To create the signature you have to take
 - algorithm specified in the header
 - encoded header
 - encoded payload
 - secret
- Signature is then used to form the third part of JWT.

```
HMACSHA256(  
    base64UrlEncode(header) + "." +  
    base64UrlEncode(payload),  
    secret  
)
```

3.2.3 Usage

Info

- User uses JWT to access protected resource by sending JWT in **Authorization** header during HTTP Request.
- If JWT is not encrypted, all information inside token is exposed to other parties (but they are unable to change it).

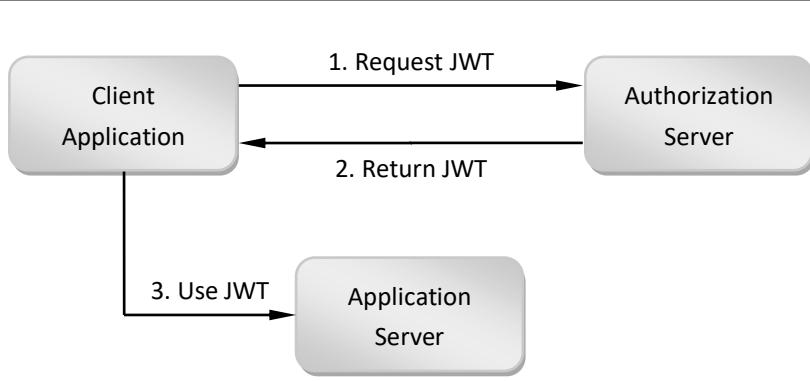
Header content

```
Authorization: Bearer <token>
```

Authorization Flow Steps

STEP	NAME	DESCRIPTION
1.	Request JWT	Client Application sends HTTP Request to Authorization Server
2.	Return JWT	Authorization Server returns JWT
3.	Use JWT	Client Application uses JWT to access restricted Resource

Authorization Flow Schema



3.2.4 Encode/Decode

Info

- This tutorial shows how to use <https://jwt.io> to
 - **Encode/Create JWT** (by entering values into Decoded Section on the right)
 - **Decode JWT** (by pasting JWT into Encoded Section on the left)
- **Algorithm** can be
 - selected from the drop down list or
 - typed directly into header
- **Secret key** can be entered with or without double quotes.
- **Signature** ignores spaces, tabs and new lines from Header and Payload JSONs (only depends on key-value pairs).

Encode JWT

- <https://jwt.io> (resulting Web Page is shown [here](#))
- Algorithm: **HS512** (select from drop down list or type into the header)
- Payload: (Copy Payload from below)
- Your-512-bit-secret: **GtRpJOGtmh29GiCcQWrvgP6mQwxI6Gsc06/nfqYIpq8oFy6ovqF79QnP9OmTBbVc54UbYRKgsvFeFgwV3uoGJw==**
- Secret base64 encoded: CHECK

Header

(based on entered Algorithm)

```
{  
  "alg" : "HS512",  
  "typ" : "JWT"  
}
```

Payload

```
{  
  "exp" : "1561852800",  
  "domain" : "some-domain",  
  "tlcIdentifier" : "some_tlc"  
}
```

Signature

(Secret key double quotes are optional. Select: secret base64 encoded)

```
HMACSHA512(  
  base64UrlEncode(header) + "." +  
  base64UrlEncode(payload),  
  GtRpJOGtmh29GiCcQWrvgP6mQwxI6Gsc06/nfqYIpq8oFy6ovqF79QnP9OmTBbVc54UbYRKgsvFeFgwV3uoGJw==  
) secret base64 encoded
```

Created JWT

(Header.Payload.Signature)

```
eyJhbGciOiJIUzUxMiIsInR5cCI6IkpXVCJ9.eyJleHAiOiE1NjE4NTI4MDAsImRvbWFpbjI6InNvbWUtZG9tYWluIiwidGxjSWRlbnRpZml1  
ciI6InNvbWUtDgxiIn0.AvhAYUMX4t7_imMd8IoSvTFRzTcAUuc5bAk9G5_kt1Eq2fPA6Faa5LLrYJ1zZ2C7XK8du_FCSzx8-wh3n0Lqaw
```