SpringBoot

---

1. Default Spring Security Authentication

Spring Dependency

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

application.properties

```
spring.security.user.name=rohal
spring.security.user.password=rohal123
spring.security.user.roles=ADMIN_ROLE
```

2. Http Basic Authentication config

InMemoryAuthentication

```
@Configuration
@EnableWebSecurity
public class SecurityConfiguration extends WebSecurityConfigurerAdapter {
  @Override
  protected void configure(AuthenticationManagerBuilder auth) throws Exception {
    auth
    .inMemoryAuthentication()
    .withUser("admin").password(passwordEncoder().encode("admin123")).roles("ADMIN")
    .and()
    .withUser("rohal").password(passwordEncoder().encode("rohal123")).roles("USER");
  }
```

```
    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http
        .authorizeRequests()
        .anyRequest().authenticated()
        .and()
        .httpBasic();
    }

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

Note : Object is created with @Beans as for Password encoder object is created

3. Configuration Role Based

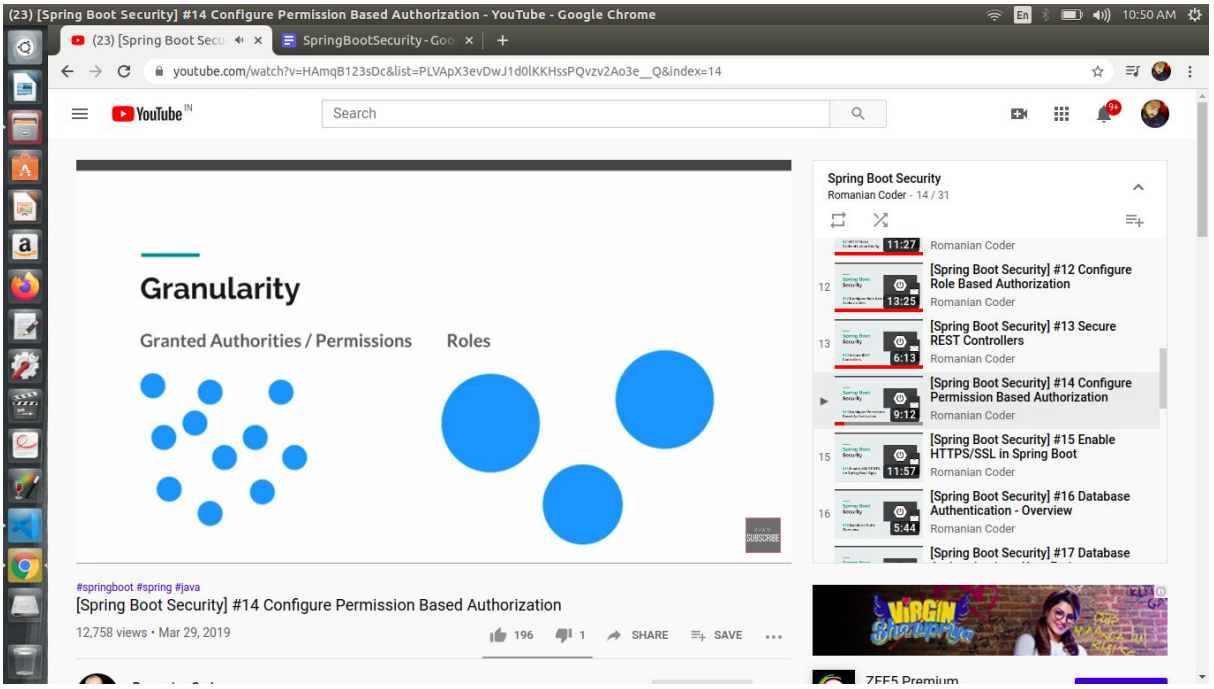
```
@Configuration
@EnableWebSecurity
public class SecurityConfiguration extends WebSecurityConfigurerAdapter {
    @Override
    protected void configure(AuthenticationManagerBuilder auth) throws Exception {
        auth
        .inMemoryAuthentication()
        .withUser("admin").password(passwordEncoder().encode("admin123")).roles("ADMIN")
        .and()
        .withUser("rohal").password(passwordEncoder().encode("rohal123")).roles("USER");
    }

    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http
        .authorizeRequests()
        .antMatchers("/public/**").permitAll()
        .antMatchers("/secure").hasAnyRole("ADMIN","USER")
        .antMatchers("/secureAdmin").hasRole("ADMIN")
        .and()
        .httpBasic();
    }
```

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

Note: Ant Matcher priority is very important as if we permit all requests then later do role based it will of nothing use.

4. Configuration Permission Based




```
@Configuration
@EnableWebSecurity
public class SecurityConfiguration extends WebSecurityConfigurerAdapter {
    @Override
    protected void configure(AuthenticationManagerBuilder auth) throws Exception {
        auth
        .inMemoryAuthentication()

.withUser("admin").password(passwordEncoder().encode("admin123")).roles("ADMIN").authorities("
ACCESS_TEST1","ACCESS_TEST2")
        .and()
```

```
.withUser("rohal").password(passwordEncoder().encode("rohal123")).roles("USER").authorities("ACC
ESS_TEST2");
  }

  @Override
  protected void configure(HttpSecurity http) throws Exception {
    http
    .authorizeRequests()
    .antMatchers("/public/**").permitAll()
    .antMatchers("/secure").hasAnyRole("ADMIN","USER")
    .antMatchers("/secureAdmin").hasRole("ADMIN")
    .antMatchers("/api/**").hasAnyAuthority("ACCESS_TEST1")
    .and()
    .httpBasic();
  }

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

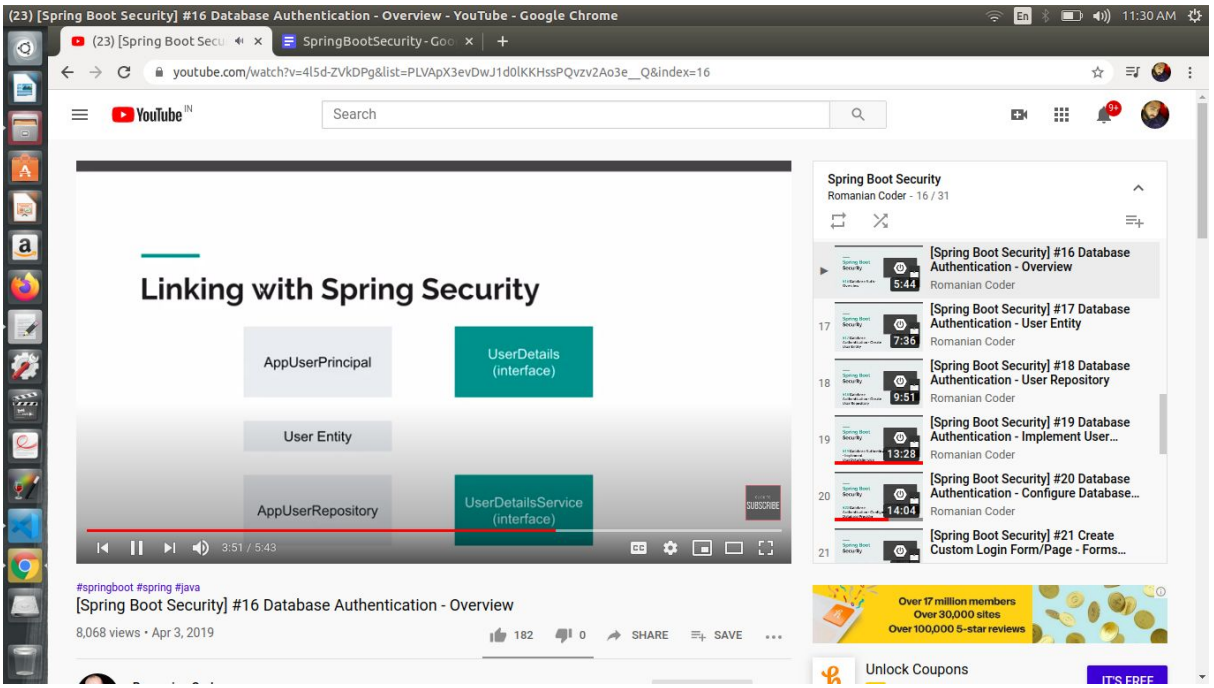
5.Enabling HTTPS/SSL Spring

Do it later

6. Database

6. Database Authentication

When we use role and authority together than role has to be added as authority in order to work perfectly



Command Line Runner

```
@Service
class DbInit implements CommandLineRunner {

@Autowired
private UserRepository userRepository;


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

       User rohal = new User("id","rohal", "rohal123", "USER","");
       User admin = new User("id","admin","admin123", "ADMIN", "ACCESS_TEST1,ACCESS_TEST2");
       rohal.setId(rohal.UniqueIdGeneration());
       admin.setId(admin.UniqueIdGeneration());

       userRepository.save(rohal);
       userRepository.save(admin);
   }

}
```

UserRepository

```
@Repository
@N1qlPrimaryIndexed
@ViewIndexed(designDoc="user",viewName="all")
public interface UserRepository extends CouchbasePagingAndSortingRepository<User, String> {

   User findByUsername(String username);
}
```

Web Security

```
@Configuration
@EnableWebSecurity
public class SecurityConfiguration extends WebSecurityConfigurerAdapter {
   @Override
   protected void configure(AuthenticationManagerBuilder auth) throws Exception {
      auth
      .inMemoryAuthentication()

.withUser("admin").password(passwordEncoder().encode("admin123")).authorities("ACCESS_TEST1","
ACCESS_TEST2","ROLE_ADMIN")
      .and()

.withUser("rohal").password(passwordEncoder().encode("rohal123")).authorities("ACCESS_TEST1","R
OLE_USER");
   }
   @Override
   protected void configure(HttpSecurity http) throws Exception {
      http
      .authorizeRequests()
      .antMatchers("/public/users").hasRole("ADMIN")
      .antMatchers("/public/**").permitAll()
      .antMatchers("/secure").hasAnyRole("ADMIN","USER")
      .antMatchers("/secureAdmin").hasRole("ADMIN")
      .antMatchers("/api/**").hasAuthority("ACCESS_TEST1")
      .and()
      .httpBasic();
   }

   @Bean
```

```
    PasswordEncoder passwordEncoder() {
        return new BCryptPasswordEncoder();
    }
}
```

Controller

```
@RestController
public class HomeController {

    @Autowired
    private UserRepository userRepository;

    @GetMapping(value = "/public/home")
    public Person getHomeApi() {
        Person person = new Person("ok", "ok", "this is home Api");
        return person;

    }

    @GetMapping(value="/public/dashboard")
    public String getDashbaordApi() {
        return "this is api for all public dashboard";
    }

    @GetMapping("/public/users")
    public Iterable<User> users() {
        return this.userRepository.findAll();
    }


}
```

User

```
@Document
public class User {

    @Id
    private String id;
```

```
@Field
private String username;

@Field
private String password;

private int active;

@Field
private String roles = "";

@Field
private String permissions = "";

public User(String cid,String username, String password, String roles, String permissions){
    this.id = cid;
    this.username = username;
    this.password = password;
    this.roles = roles;
    this.permissions = permissions;
    this.active = 1;
}

protected User(){}

public String getId() {
    return id;
}

public String getUsername() {
    return username;
}

public String getPassword() {
    return password;
}

public int getActive() {
    return active;
}

public String getRoles() {
    return roles;
}
```

```
    public String getPermissions() {
        return permissions;
    }

    public List<String> getRoleList(){
        if(this.roles.length() > 0){
            return Arrays.asList(this.roles.split(","));
        }
        return new ArrayList<>();
    }

    public List<String> getPermissionList(){
        if(this.permissions.length() > 0){
            return Arrays.asList(this.permissions.split(","));
        }
        return new ArrayList<>();
    }

    public String UniqueIdGeneration() {
        return  UUID.randomUUID().toString();
    }

    public void setId(String id) {
        this.id = id;
    }
}
```

Implementing UserDetailService

```
public class UserPrincipal implements UserDetails {

    /**
     *
     */
    private static final long serialVersionUID = 1L;

    User user;

    public UserPrincipal(User user) {
        this.user = user;
            }

            @Override
```

```
public Collection<? extends GrantedAuthority> getAuthorities() {
    List<GrantedAuthority> authorities = new ArrayList<>();

    // Extracting list of permissios (name)
    this.user.getPermissionList().forEach( p -> {
        GrantedAuthority authority = new SimpleGrantedAuthority(p);
        authorities.add(authority);
    });

    // Extracting list of roles (ROLE_name)

    this.user.getRoleList().forEach( p -> {
        GrantedAuthority authority = new SimpleGrantedAuthority("ROLE_"+p);
        authorities.add(authority);
    });
    return authorities;
}

@Override
public String getPassword() {
    return this.user.getPassword();
}

@Override
public String getUsername() {
    return this.user.getUsername();
}

@Override
public boolean isAccountNonExpired() {
    return true;
}

@Override
public boolean isAccountNonLocked() {
    return false;
}

@Override
public boolean isCredentialsNonExpired() {
    return true;
}

@Override
public boolean isEnabled() {
    return this.user.getActive() == 1;
```

```
    }

}
```

UserPrincipalDetailService

```
public class UserPrincipalDetailService implements UserDetailsService {

    @Autowired
    private UserRepository userRepository;

    @Override
    public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {
      User user = userRepository.findByUsername(username);
      UserPrincipal userPrincipal = new UserPrincipal(user);
      return userPrincipal;
    }

}
```

UserPrincipal

```
public class UserPrincipal implements UserDetails {

    /**
     *
     */
    private static final long serialVersionUID = 1L;

    private User user;

    public UserPrincipal(User user){
        this.user = user;
    }

    @Override
    public Collection<? extends GrantedAuthority> getAuthorities() {
```

```
    List<GrantedAuthority> authorities = new ArrayList<>();

    // Extract list of permissions (name)
    this.user.getPermissionList().forEach(p -> {
        GrantedAuthority authority = new SimpleGrantedAuthority(p);
        authorities.add(authority);
    });

    // Extract list of roles (ROLE_name)
    this.user.getRoleList().forEach(r -> {
        GrantedAuthority authority = new SimpleGrantedAuthority("ROLE_" + r);
        authorities.add(authority);
    });

    return authorities;
}

@Override
public String getPassword() {
    return this.user.getPassword();
}

@Override
public String getUsername() {
    return this.user.getUsername();
}

@Override
public boolean isAccountNonExpired() {
    return true;
}

@Override
public boolean isAccountNonLocked() {
    return true;
}

@Override
public boolean isCredentialsNonExpired() {
    return true;
}

@Override
public boolean isEnabled() {
    return this.user.getActive() == 1;
}
```

```
}
```

LDAP Configuration with Database

WebConfiguration Class

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

 @Autowired
 MyAuthoritiesPopulator myUserDetailService;

 @Autowired
 CustomUserDetailsMapper mycustomeDetailMapper;

 @Autowired
 UserService myUserDetailServicerem;

  @Override
  protected void configure(HttpSecurity http) throws Exception {
    http
      .authorizeRequests()
      .anyRequest().authenticated()
      .and()
      .formLogin()
      .loginPage("/login")
      .usernameParameter("user")
      .passwordParameter("pwd")
      .defaultSuccessUrl("/dashboard", true)
      .failureForwardUrl("/login")
      .permitAll()
      .and()
      .logout()
      .logoutSuccessUrl("/")
      .and()
      .sessionManagement().sessionCreationPolicy(SessionCreationPolicy.NEVER).and()
      .rememberMe().tokenValiditySeconds(86400);
  }

  @Override
```

```
    public void configure(AuthenticationManagerBuilder auth) throws Exception {
      auth.
      ldapAuthentication()
         .userDetailsContextMapper(mycustomeDetailMapper)
         .userDnPatterns("uid={0},ou=people")
         .groupSearchBase("ou=groups")
         .contextSource()
          .url("ldap://localhost:8389/dc=springframework,dc=org")
          .and()
         .passwordCompare()
          .passwordEncoder(new BCryptPasswordEncoder())
          .passwordAttribute("userPassword")
          .and().ldapAuthoritiesPopulator(myUserDetailService)
          .and().userDetailsService(myUserDetailServicerem);
    }

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

LdapAuthoritiesPopulator Class

```
@Service
public class MyAuthoritiesPopulator implements LdapAuthoritiesPopulator {

    @Autowired
    private UserRepository userRepository;

    @Override
    public Collection<? extends GrantedAuthority> getGrantedAuthorities(DirContextOperations
userData,String username) {
        System.out.println("Name is "+username);
        User user = userRepository.findByUsername(username);
        List<GrantedAuthority> authorities = new ArrayList<>();

        // Extract list of permissions  (name)
        user.getPermissionList().forEach(p -> {
            GrantedAuthority authority = new SimpleGrantedAuthority(p);
            authorities.add(authority);
```

```
        });

        // Extract list of roles (ROLE_name)
        user.getRoleList().forEach(r -> {
            GrantedAuthority authority = new SimpleGrantedAuthority("ROLE_" + r);
            authorities.add(authority);
        });

        return authorities;
    }

}
```

LdapUserDetailsMapper Class

```
@Service
public class CustomUserDetailsMapper  extends LdapUserDetailsMapper {

    // @Autowired
    // private UserDetailsService userDetailService;

    @Autowired
    private UserRepository userRepository;


    @Override
    public UserDetails mapUserFromContext(DirContextOperations ctx, String username,
            Collection<? extends GrantedAuthority> authorities) {
            System.out.println("Name is"+username);
            //return (UserDetails) this.userDetailService.loadUserByUsername(username);

            User user = userRepository.findByUsername(username);
            UserPrincipal userPrincipal = new UserPrincipal(user);
            return userPrincipal;

    }
}
```

UserDetailsService Class

```
@Service
public class UserService implements UserDetailsService {

    @Autowired
    private UserRepository userRepository;

    @Override
    public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {
        //User user = new User("rohal","rohal123", new ArrayList<>());
        //UserPrincipal userPrincipal = new UserPrincipal(user);
        //return userPrincipal;
        User user = userRepository.findByUsername(username);
        UserPrincipal userPrincipal = new UserPrincipal(user);
        return userPrincipal;
    }

}
```

UserDetails Class

```
public class UserPrincipal implements UserDetails {

    /**
     *
     */
    private static final long serialVersionUID = 1L;

    private User user;

    public UserPrincipal(User user){
        this.user = user;
    }

    @Override
    public Collection<? extends GrantedAuthority> getAuthorities() {
        List<GrantedAuthority> authorities = new ArrayList<>();
```

```
        // Extract list of permissions (name)
        this.user.getPermissionList().forEach(p -> {
            GrantedAuthority authority = new SimpleGrantedAuthority(p);
            authorities.add(authority);
        });

        // Extract list of roles (ROLE_name)
        this.user.getRoleList().forEach(r -> {
            GrantedAuthority authority = new SimpleGrantedAuthority("ROLE_" + r);
            authorities.add(authority);
        });

        return authorities;
    }

    @Override
    public String getPassword() {
        return this.user.getPassword();
    }

    @Override
    public String getUsername() {
        return this.user.getUsername();
    }

    @Override
    public boolean isAccountNonExpired() {
        return true;
    }

    @Override
    public boolean isAccountNonLocked() {
        return true;
    }

    @Override
    public boolean isCredentialsNonExpired() {
        return true;
    }

    @Override
    public boolean isEnabled() {
        return this.user.getActive() == 1;
    }
}
```

UserRepository Class

```
@Repository
@N1qlPrimaryIndexed
@ViewIndexed(designDoc="user",viewName="all")
public interface UserRepository extends CouchbasePagingAndSortingRepository<User, String> {
   User findByUsername(String username);
}
```

User Class

```
@Document
public class User {

   @Id
   private String id;

   @Field
   private String username;

   @Field
   private String password;

   private int active;

   @Field
   private String roles = "";

   @Field
   private String permissions = "";

   public User(String cid,String username, String password, String roles, String permissions){
      this.id = cid;
      this.username = username;
      this.password = password;
      this.roles = roles;
      this.permissions = permissions;
      this.active = 1;
   }
```

```
protected User(){}

public String getId() {
    return id;
}

public String getUsername() {
    return username;
}

public String getPassword() {
    return password;
}

public int getActive() {
    return active;
}

public String getRoles() {
    return roles;
}

public String getPermissions() {
    return permissions;
}

public List<String> getRoleList(){
    if(this.roles.length() > 0){
        return Arrays.asList(this.roles.split(","));
    }
    return new ArrayList<>();
}

public List<String> getPermissionList(){
    if(this.permissions.length() > 0){
        return Arrays.asList(this.permissions.split(","));
    }
    return new ArrayList<>();
}

public String UniqueIdGeneration() {
    return  UUID.randomUUID().toString();
}

public void setId(String id) {
```

```
        this.id = id;
    }
}
```

CommandLine Runner Class

```
@Service
class DbInit implements CommandLineRunner {

@Autowired
private UserRepository userRepository;

@Autowired
private PasswordEncoder passwordEncoder;


@Override
    public void run(String... args) throws Exception {
        //userRepository.deleteAll();


        User rohal = new User("id","ben", passwordEncoder.encode("benspassword"), "USER","");
        //User admin = new User("id","admin",passwordEncoder.encode("admin123"), "ADMIN",
"ACCESS_TEST1,ACCESS_TEST2");
        rohal.setId(rohal.UniqueIdGeneration());
        //admin.setId(admin.UniqueIdGeneration());
       userRepository.save(rohal);
       //User user = userRepository.findByUsername("bens");
       //userRepository.save(admin);
       // //List<User> list = Arrays.asList(rohal,admin);
       // //userRepository.saveAll(list);
       // userRepository.save(rohal);
       // userRepository.save(admin);
       //User user=userRepository.findByUsername("rohal");
       //System.out.println(user);
    }
    @Bean
PasswordEncoder passwordEncoder() {
return new BCryptPasswordEncoder();
}

}
```