

# Using Spring Security Reactive

Zoltan Altfatter

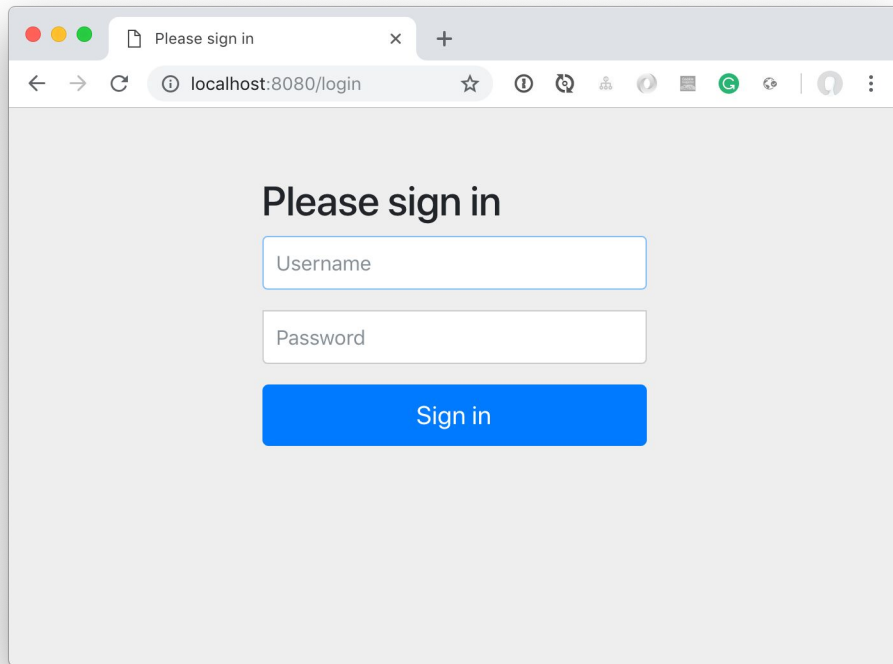
# Spring Security reactive support

- Optional reactive
- if there is **spring-boot-starter-webflux** without **spring-boot-starter-web** and with the **spring-security-starter-security** will configure the reactive spring security
- if there is **spring-boot-starter-web** and spring-security-starter-security then it will configure non-reactive spring security

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

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

# New built-in login screen



A screenshot of a web browser window displaying a login page. The browser's address bar shows the URL `localhost:8080/login`. The page has a light gray background and contains the following elements:

- A heading "Please sign in" in bold black text.
- A text input field labeled "Username" with a light blue border.
- A text input field labeled "Password" with a light gray border.
- A blue rectangular button labeled "Sign in" in white text.

# Technical Username & Password

Default username is **user** but configurable with the following property:

```
spring.security.user.user=user
```

Default password from the logs:

```
2018-10-22 19:19:34.465 INFO 53449 --- [ restartedMain]
ctiveUserDetailsServiceAutoConfiguration :
```

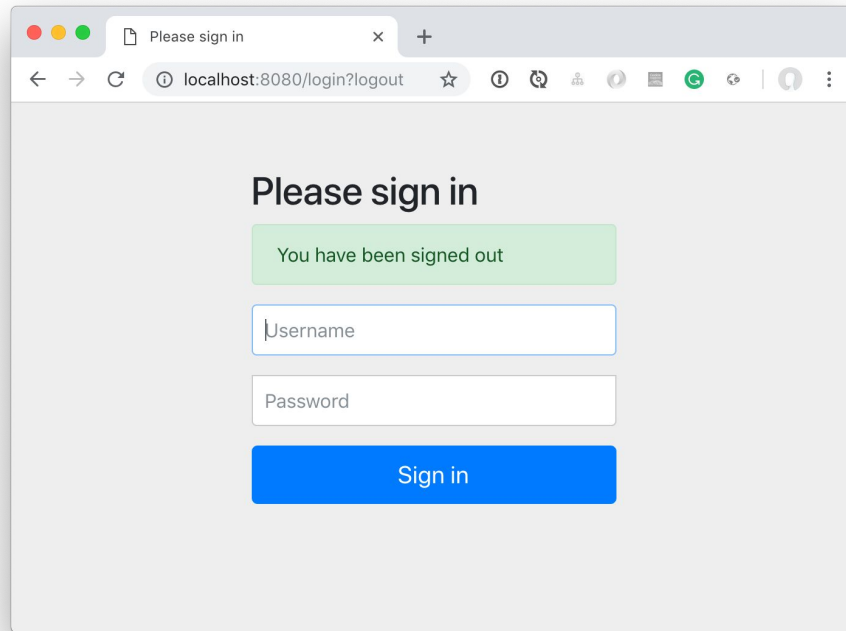
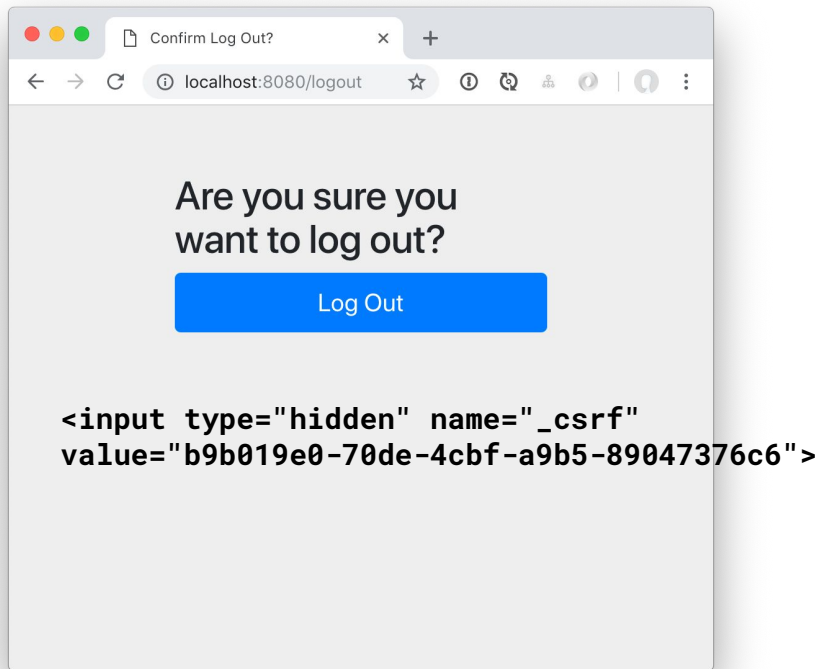
**Using generated security password: d3b0a710-aac1-49e3-b6de-59741fa3a52e**

```
2018-10-22 19:19:35.481 INFO 53449 --- [ restartedMain]
s.w.r.r.m.a.RequestMappingHandlerMapping : Mapped "{[/],methods=[GET]}"
onto java.lang.String sample.IndexController.index()
```

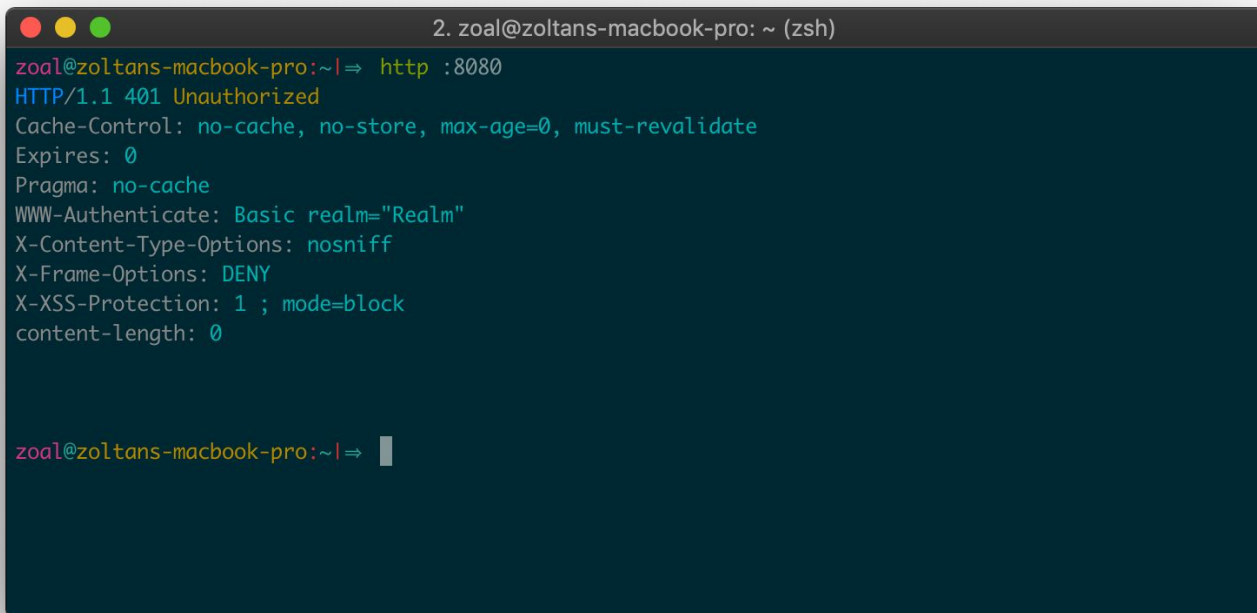
**or configuration**

```
spring.security.user.password=secret
```

# New built-in logout page



# Content negotiation



```
2. zoal@zoltans-macbook-pro: ~ (zsh)
zoal@zoltans-macbook-pro:~|⇒ http :8080
HTTP/1.1 401 Unauthorized
Cache-Control: no-cache, no-store, max-age=0, must-revalidate
Expires: 0
Pragma: no-cache
WWW-Authenticate: Basic realm="Realm"
X-Content-Type-Options: nosniff
X-Frame-Options: DENY
X-XSS-Protection: 1 ; mode=block
content-length: 0

zoal@zoltans-macbook-pro:~|⇒
```

The image shows a terminal window with a dark blue background. At the top, the window title is "2. zoal@zoltans-macbook-pro: ~ (zsh)". The terminal content shows a user prompt "zoal@zoltans-macbook-pro:~|⇒" followed by the command "http :8080". The response is "HTTP/1.1 401 Unauthorized". Below this, several HTTP headers are listed: "Cache-Control: no-cache, no-store, max-age=0, must-revalidate", "Expires: 0", "Pragma: no-cache", "WWW-Authenticate: Basic realm=\"Realm\"", "X-Content-Type-Options: nosniff", "X-Frame-Options: DENY", "X-XSS-Protection: 1 ; mode=block", and "content-length: 0". At the bottom, the prompt "zoal@zoltans-macbook-pro:~|⇒" is shown again with a cursor.

# MapReactiveUserDetailsService

`@EnableWebFluxSecurity`

```
public class SecurityConfig {
```

`@Bean`

```
    MapReactiveUserDetailsService userDetailsService() {  
        UserDetails rob = User.withUsername("user@example.com")  
            .password("password")  
            .roles("USER")  
            .build();  
        return new MapReactiveUserDetailsService(rob);  
    }  
}
```

- **EnableWebFluxSecurity** is not needed anymore because Spring Boot has auto configuration for it, see **WebFluxSecurityAutoConfiguration**

# ReactiveUserDetailsService

```
@Component
public class RepositoryReactiveUserDetailsService implements ReactiveUserDetailsService {

    private final UserRepository users;

    public RepositoryReactiveUserDetailsService(UserRepository users) {
        this.users = users;
    }

    @Override
    public Mono<UserDetails> findByUsername(String username) {
        return this.users.findByEmail(username).map(CustomUserDetails::new);
    }
}

class CustomUserDetails extends User implements UserDetails {

    public CustomUserDetails(User user) {
        super(user);
    }

    @Override
    public Collection<? extends GrantedAuthority> getAuthorities() {
        return AuthorityUtils.createAuthorityList("ROLE_USERR");
    }

    @Override
    public String getUsername() {
        return getEmail();
    }
    ...
}
```



# Custom user storage

```
@Document
public class User {
```

```
    @Id
    private Long id;
```

```
    @Indexed
    @NotEmpty(message = "This field is required")
    private String email;
```

```
    @NotEmpty(message = "This field is required")
    private String password;
```

```
    @NotEmpty(message = "This field is required")
    private String firstName;
```

```
    @NotEmpty(message = "This field is required")
    private String lastName;
```

```
    ...
}
```

```
@ModelAttribute("currentUser")
Mono<User> currentUser(@AuthenticationPrincipal Mono<User> principal) {
    return principal;
}
```

```
th:text="${currentUser.firstName}"
```

```
@Target({ ElementType.PARAMETER, ElementType.TYPE })
@Retention(RetentionPolicy.RUNTIME)
@Documented
@AuthenticationPrincipal
public @interface CurrentUser {
}
```

```
@ModelAttribute("currentUser")
Mono<User> currentUser(@CurrentUser Mono<User> currentUser) {
    return currentUser;
}
```

# DelegatingPasswordEncoder

- A **PasswordEncoder** that delegates to another **PasswordEncoder** based on a **prefixed identifier**

```
Map<String, PasswordEncoder> encoders = new HashMap<>();  
encoders.put("bcrypt", new BCryptPasswordEncoder());  
encoders.put("noop", NoOpPasswordEncoder.getInstance());  
encoders.put("pbkdf2", new Pbkdf2PasswordEncoder());  
encoders.put("scrypt", new SCryptPasswordEncoder());  
encoders.put("sha256", new StandardPasswordEncoder());
```

- **password encoding storage** evolves
- Spring Security cannot be changing all the time how it stores the passwords

@Bean

```
PasswordEncoder passwordEncoder() {  
    return PasswordEncoderFactories.createDelegatingPasswordEncoder();  
}
```

# Password in bcrypt

```
@EnableWebFluxSecurity
public class SecurityConfig {

    @Bean
    MapReactiveUserDetailsService userDetailsService() {
        UserDetails rob = User.withUsername("user@example.com")
            .password("{bcrypt}$2a$12$3EJ5urLe4tsqYBDGWH548uZHcu.rcKnnkDoAvZTc")
            .roles("USER")
            .build();
        return new MapReactiveUserDetailsService(rob);
    }
}
```

# SecurityControllerAdvice

```
@ControllerAdvice
public class SecurityControllerAdvice {

    @ModelAttribute
    Mono<CsrfToken> csrfToken(ServerWebExchange exchange) {
        Mono<CsrfToken> csrfToken = exchange.getAttribute(CsrfToken.class.getName());
        return csrfToken.doOnSuccess(token -> exchange.getAttributes()
            .put(CsrfRequestDataValueProcessor.DEFAULT_CSRF_ATTR_NAME, token));
    }

    @ModelAttribute("currentUser")
    Mono<Principal> currentUser(Mono<Principal> principal) {
        return principal;
    }
}
```

# Minimal configuration

@Bean

```
public SecurityWebFilterChain springSecurityFilterChain(ServerHttpSecurity http) {  
    http  
        .authorizeExchange()  
            .anyExchange().authenticated()  
        .and()  
            .httpBasic()  
        .and()  
            .formLogin();  
    return http.build();  
}
```

# Custom login page

@Bean

```
SecurityWebFilterChain springSecurityFilterChain(ServerHttpSecurity http) {  
    http  
        .authorizeExchange()  
            .pathMatchers("/login", "/signup", "/webjars/**").permitAll()  
            .anyExchange().authenticated()  
            .and()  
        .httpBasic().and()  
        .formLogin()  
            .loginPage("/login");  
    return http.build();  
}
```

# Spring Boot CLI

- `sdk install springboot`
- `brew install pivotal/tap/springboot`

**`spring encodepassword [options] <password to encode>`**

Encode a password for use with Spring Security

# Update password storage

@Component

```
class RepositoryReactiveUserDetailsService implements ReactiveUserDetailsService, ReactiveUserDetailsPasswordService {  
  
    private final UserRepository users;  
  
    public RepositoryReactiveUserDetailsService(UserRepository users) {  
        this.users = users;  
    }  
  
    @Override  
    public Mono<UserDetails> findByUsername(String username) {  
        return this.users.findByEmail(username).map(CustomUser::new);  
    }  
  
    @Override  
    public Mono<UserDetails> updatePassword(UserDetails user, String newPassword) {  
        return this.users.findByEmail(user.getUsername())  
            .doOnSuccess(u -> u.setPassword(newPassword))  
            .flatMap(this.users::save)  
            .map(CustomUser::new);  
    }  
}
```



# SignupController

@Controller

```
public class SignupController {  
  
    private final UserRepository users;  
    private final PasswordEncoder encoder;  
  
    public SignupController(UserRepository users, PasswordEncoder encoder) {  
        this.users = users;  
        this.encoder = encoder;  
    }  
  
    @GetMapping("/signup")  
    public Mono<String> signupForm(@ModelAttribute User user) {  
        return Mono.just("signup/form");  
    }  
  
    @PostMapping("signup")  
    public Mono<String> signup(@Valid User user, BindingResult result) {  
        if(result.hasErrors()) {  
            return signupForm(user);  
        }  
        return Mono.just(user)  
            .publishOn(Schedulers.parallel())  
            .doOnNext(u -> u.setPassword(this.encoder.encode(u.getPassword())))  
            .flatMap(this.users::save)  
            .then(Mono.just("redirect:/"));  
    }  
}
```

# Authorization ServerHttpSecurity

@Configuration

```
public class SecurityConfig {
```

@Bean

```
SecurityWebFilterChain springSecurityFilterChain(ServerHttpSecurity http) {  
    http
```

```
        .authorizeExchange()  
        .pathMatchers("/users").access(this::isRob)  
        .pathMatchers("/login", "/signup", "/webjars/**").permitAll()  
        .anyExchange().authenticated()  
        .and()  
        .httpBasic().and()  
        .formLogin()  
        .loginPage("/login");
```

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

```
}
```

```
private Mono<AuthorizationDecision> isRob(Mono<Authentication> authentication, AuthorizationContext context) {
```

```
    return authentication  
        .map(Authentication::getName)  
        .map(username -> username.startsWith("rob@"))  
        .map(AuthorizationDecision::new);
```

```
}
```

```
}
```

# @EnableReactiveMethodSecurity

```
@Service
public class HelloWorldMessageService {

    @PreAuthorize("hasRole('ADMIN')")
    public Mono<String> findMessage() {
        return Mono.just("Hello World!");
    }
}

@Configuration
@EnableReactiveMethodSecurity
public class SecurityConfig {
    ...
}
```

- **findMessage** is only invoked by a user with a role ADMIN
- Currently only expressions with return type **Boolean** or **boolean** are supported (which do not block)