

SECURITY CONFIGURATION

```
@Configuration
@EnableMethodSecurity(securedEnabled = true)
public class SecurityConfig {

    // Miel Verkerken
    @Bean
    public DataSource datasource() {...};

    // Configuration for jdbc authentication
    // Miel Verkerken +1
    @Bean
    public UserDetailsManager users(DataSource datasource) {...}

    // mverkerk +1
    @Bean
    public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {...}
}
```

Datasource for saving credentials

Configure credentials using
UserDetailsManager

Configure the filterChain

AUTHORIZING URL

- Define specific authorization restrictions for URLs
- Support “Ant-style” pattern matching
 - `"/admin/*"` only matches `"/admin/xxx"`
 - `"/admin/**"` matches any path under `/admin`
 - Such as `"/admin/database/access-control"`

```
@Bean
public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
    return http.authorizeHttpRequests(requests -> requests
        .requestMatchers(new AntPathRequestMatcher("/admin.html")).hasRole("ADMIN"))
        .build();
}
```

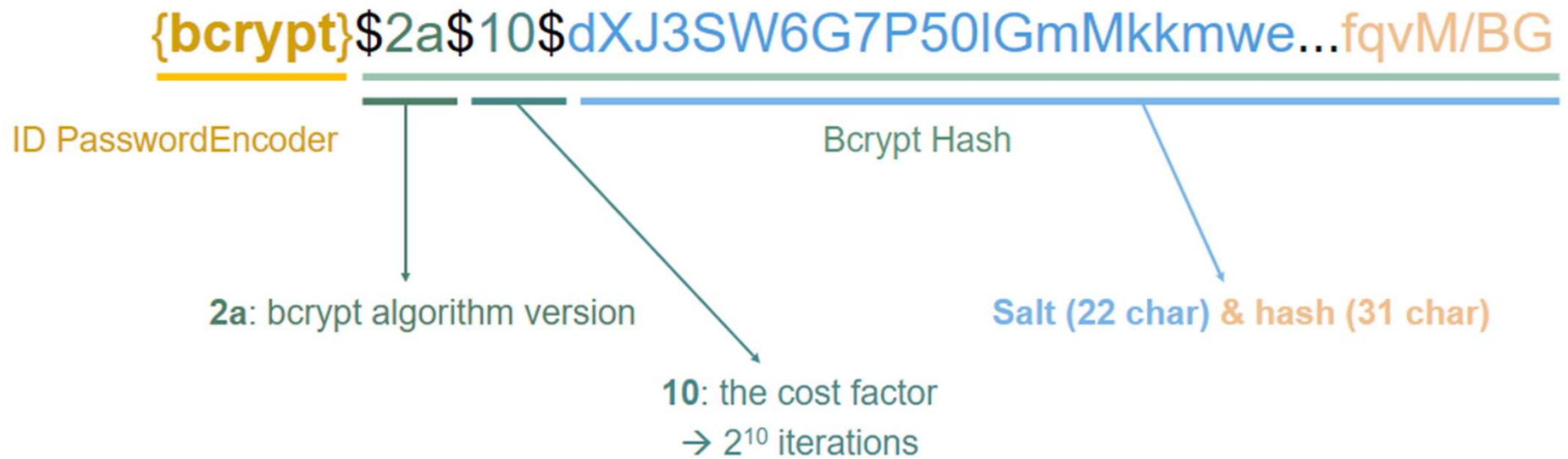
Match *all* URLs starting with
`/admin` (ANT-style path)

User must have
ADMIN role

CHALLENGES PASSWORD ENCODING (1)

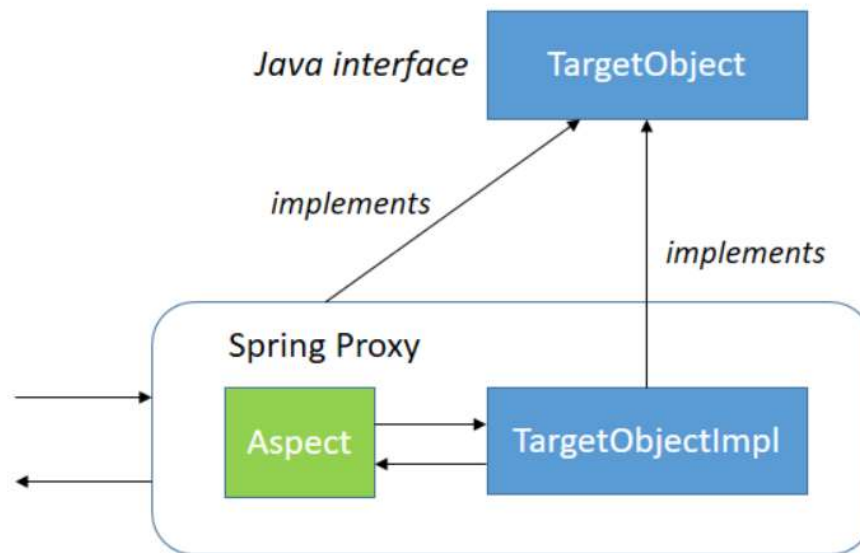
- Should be future proof
 - Secure today is not secure tomorrow
 - New encoding schemes emerge in the future
- Should accommodate old password formats
 - Co-exist new and old format passwords
- DelegatingPasswordEncoder
 - {id}encodedPassword
 - {**bcrypt**}\$2a\$10\$dXJ3SW6G7P50IGmMkkmwe...fqvM/BG

CHALLENGES PASSWORD ENCODING (2)

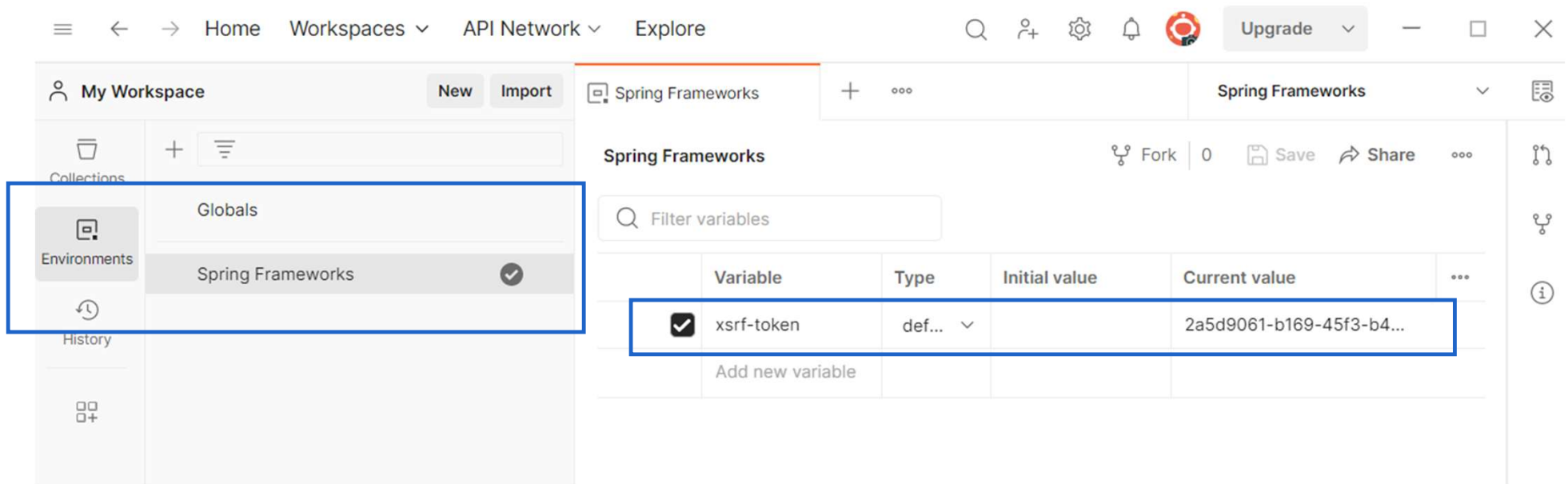


METHOD SECURITY

- Spring Security uses AOP for method-level security
 - Annotations: either Spring's own or JSR-250
- Spring uses a AOP Proxy



CSRF TOKEN IN POSTMAN (1)



The screenshot shows the Postman application interface. On the left sidebar, under 'My Workspace', the 'Environments' section is expanded, and 'Spring Frameworks' is selected. The main panel displays the 'Spring Frameworks' environment variables. A table lists the variables, with 'xsrf-token' highlighted. The table has columns for 'Variable', 'Type', 'Initial value', and 'Current value'.

Variable	Type	Initial value	Current value
<input checked="" type="checkbox"/> xsrf-token	def... ▾		2a5d9061-b169-45f3-b4...
Add new variable			

CSRF TOKEN IN POSTMAN (2)

The screenshot shows the Postman interface with a workspace named 'My Workspace'. On the left sidebar, under 'Collections', the 'Spring Frameworks' collection is expanded, showing a list of endpoints. The 'POST create post' endpoint is selected. The main panel shows the request configuration for 'POST create post' at the URL 'http://localhost:8080/posts'. The 'Headers' tab is active, showing a table with one header: 'X-XSRF-TOKEN' with a value of '{{xsrf-token}}'. The 'Send' button is visible on the right.

Spring Frameworks / POST create post

POST http://localhost:8080/posts

Params Auth Headers (11) Body Pre-req. Tests Settings Cookies

Headers 10 hidden

	Key	Value	D...	...	Bulk Edit	Presets
<input checked="" type="checkbox"/>	X-XSRF-TOKEN	{{xsrf-token}}				
	Key	Value	Description			

CSRF TOKEN IN POSTMAN (3)

The screenshot displays the Postman application interface. On the left, the 'My Workspace' sidebar shows a collection named 'Spring Frameworks' containing several API endpoints. The 'POST create post' endpoint is selected. The main panel shows the request configuration for this endpoint, including the method 'POST' and the URL 'http://localhost:8080/posts'. The 'Tests' tab is active, showing a JavaScript script to extract the XSRF token from the response and store it in an environment variable.

My Workspace **New** **Import** Spring Frameworks **POST POST create post** Spring Frameworks

Spring Frameworks / **POST create post** Save Send

POST http://localhost:8080/posts

Params Auth Headers (11) Body Pre-req **Tests** Settings Cookies

```
1 var xsrfCookie = postman.getResponseCookie('XSRF-TOKEN');  
2 postman.setEnvironmentVariable("xsrf-token", xsrfCookie.value)
```


ZOEK FUNCTIONALITEIT

```
/**
 * Search for blogPost based on keywords
 */
@GetMapping("/posts/search")
public List<BlogPost> searchPosts(@RequestParam(name = "q") String keyword) {
    return postDAO.searchPostsByTitleContaining(keyword);
}
```

Controller

DAO

```
@Override
public List<BlogPost> searchPostsByTitleContaining(String keyword) {
    return repository.findByTitleContaining(keyword);
}
```

```
public interface BlogPostRepository extends JpaRepository<BlogPost, Long> {
    List<BlogPost> findByTitleContaining(String keyword);
}
```

Repository