Spring Security

Authentication and Authorization
Custom Annotation

Authentication Types

1. Basic Authentication
2. Form Login

Default Security Setup

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

Default Security Configuration

- By default, authentication gets enabled for the application.
- Content negotiation is used to determine if **http basic** or **form login** should be used

Verification

1. Postman - http://localhost:8080/api/users/1
2. Browser - http://localhost:8080/api/users/1

Default Credentials

Username: user

Password: Using generated security password: 1d61adfb-bc81-4fe7-a366-420d972131d1

Custom Username and Password

spring.security.user.name = **admin**
spring.security.user.password = **password**

Disable Autoconfiguration

To discard the security auto-configuration and add our own configuration, we need to exclude the *SecurityAutoConfiguration* class.

```
@SpringBootApplication(exclude = SecurityAutoConfiguration.class)
@EnableSwagger2
public class MoneyTransferAppApplication {

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

Disabling vs. Surpassing Security Auto-Configuration

Disabling: it's just like adding the spring security dependency and the whole setup from scratch. This can be useful in several cases:

1. Integrating application security with a custom security provider
2. Migrating a legacy Spring application with already existing security setup – to Spring Boot.

Surpassing: can be achieved by adding in our new/custom configuration classes. This is typically easier, as we're just customizing an existing security setup to fulfill our needs.

Configuring Spring Boot Security

```
@Configuration
@EnableWebSecurity
public class BasicConfiguration extends WebSecurityConfigurerAdapter {
  @Override
  protected void configure(AuthenticationManagerBuilder auth) throws Exception {
    auth
        .inMemoryAuthentication()
        .withUser("user").password("{noop}password").roles("USER")
        .and()
        .withUser("admin").password("{noop}password").roles("USER", "ADMIN");
  }
  @Override
  protected void configure(HttpSecurity http) throws Exception {
    http
        .authorizeRequests()
        .anyRequest()
        .authenticated()
        .and()
        .httpBasic();
  }
}
```

@EnableWebSecurity **annotation is crucial if we disable the default security configuration**

Cover Basic Security Configuration with Integration Test

```
@Before
public void setUp() throws MalformedURLException {
   restTemplate = new TestRestTemplate("user", "password");
   base = new URL("http://localhost:" + port);
}

@Test
public void whenLoggedUserRequestsHomePage_ThenSuccess()
      throws IllegalStateException, IOException {
   ResponseEntity<String> response
         = restTemplate.getForEntity(base.toString(), String.class);

   assertEquals(HttpStatus.OK, response.getStatusCode());
   assertTrue(response.getBody().contains("Welcome Home!"));
}
```

Password Encoders

```
public class PasswordEncoderFactories {
  public static PasswordEncoder createDelegatingPasswordEncoder() {
     String encodingId = "bcrypt";
     Map<String, PasswordEncoder> encoders = new HashMap();
     encoders.put(encodingId, new BCryptPasswordEncoder());
     encoders.put("ldap", new LdapShaPasswordEncoder());
     encoders.put("MD4", new Md4PasswordEncoder());
     encoders.put("MD5", new MessageDigestPasswordEncoder("MD5"));
     encoders.put("noop", NoOpPasswordEncoder.getInstance());
     encoders.put("pbkdf2", new Pbkdf2PasswordEncoder());
     encoders.put("scrypt", new SCryptPasswordEncoder());
     encoders.put("SHA-1", new MessageDigestPasswordEncoder("SHA-1"));
     encoders.put("SHA-256", new MessageDigestPasswordEncoder("SHA-256"));
     encoders.put("sha256", new StandardPasswordEncoder());
     return new DelegatingPasswordEncoder(encodingId, encoders);
  }

  private PasswordEncoderFactories() {
  }
}
```

Password Storage Format

1. **{bcrypt}**$2a$10$dXJ3SW6G7P50lGmMkkmwe.20cQQubK3.HZWzG3YB1tlRy.fqvM/BG
2. **{noop}**password
3. **{pbkdf2}**5d923b44a6d129f3ddf3e3c8d29412723dcbde72445e8ef6bf3b508fbf17fa4ed4d6b99ca763d8dc
4. **{scrypt}**$e0801$8bWJaSu2IKSn9Z9kM+TPXfOc/9bdYSrN1oD9qfVThWEwdRTnO7re7Ei+fUZRJ68k9lTyuTeUp4of4g24hHnazw==$OAOec05+bXxvuu/1qZ6NUR+xQYvYv7BeL1QxwRpY5Pc=
5. **{sha256}**97cde38028ad898ebc02e690819fa220e88c62e0699403e94fff291cfffaf8410849f27605abcbc0

General Password Format

```
{id}encodedPassword
```

"id" is an identifier used to look up which `PasswordEncoder` should be used and "encodedPassword" is the original encoded password for the selected `PasswordEncoder`. The "id" must be at the beginning of the password, start with "{" and end with "}". If the "id" cannot be found, the "id" will be null. For example, the following might be a list of passwords encoded using different "id". All of the original passwords are "password".

Spring Method Security

1. Spring Security supports authorization semantics at the method level

Enabling Method Security 1

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

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

Enabling Method Security 2

```
@Configuration
@EnableGlobalMethodSecurity(
      prePostEnabled = true,
      securedEnabled = true,
      jsr250Enabled = true)
public class MethodSecurityConfig
      extends GlobalMethodSecurityConfiguration {
}
```

- The *prePostEnabled* property enables Spring Security pre/post annotations
- The *securedEnabled* property determines if the *@Secured* annotation should be enabled
- The *jsr250Enabled* property allows us to use the *@RoleAllowed* annotation

@PreAuthorize

```
//provide expression-based access control
@GetMapping("/")
@PreAuthorize("hasRole('ROLE_ADMIN')")
public ResponseEntity<Iterable<User>> index() {
   return ResponseEntity.ok()
          .header("Request-Id", "request-id")
          .body(userRepository.findAll());
}
```

@PostAuthorize

```
@ApiOperation(value = "find one user by id")
@GetMapping("/{id}")
@PostAuthorize("returnObject.getBody().firstName =='Akyl'")
public ResponseEntity<User> findOne(@PathVariable Long id) {
   User user = userRepository.findById(id).get();
   return new ResponseEntity<>(user, HttpStatus.OK);
}
```

Custom Annotation - Method Security Meta-Annotation

```
@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.METHOD)
@PreAuthorize("hasRole('SUPER_ADMIN')")
public @interface IsSuperAdmin {
}
```

```
@GetMapping("/")
@IsSuperAdmin
public ResponseEntity<Iterable<Transaction>> findAll() {
    Iterable<Transaction> transactions = transactionRepository.findAll();
    if (transactions.iterator().hasNext()) {
        return ResponseEntity.ok(transactions);
    }
    return ResponseEntity.notFound().build();
}
```

```
.withUser("superadmin").password("{noop}password").roles("USER", "ADMIN", "SUPER_ADMIN");
```

Security Annotation at the Class Level

```
@Service("HomeService")
@PreAuthorize("hasRole('ROLE_ADMIN')")
public class HomeService {

   public String welcome()
   {
      return  "Welcome Home!";
   }
}
```

@PostFilter and @PreFilter Annotations

```
public interface TransactionRepository extends CrudRepository<Transaction,
Long> {
   Iterable<Transaction> findByUserId(Long id);

   @PostFilter("filterObject.transactionId == authentication.name")
   List<Transaction> findAllByAmount(Long amount);
}
```

Exercises

1. Explore @PreFilter and write one example
2. Secure delete user with @RolesAllowed or @Secured
3. Create create-transaction endpoint
    a. Write end-to-end test
    b. Update documentation
        i. @ApiOperation
    c. Secure it with role ROLE_ROOT
4. Create delete-transaction endpoint
    a. Write end-to-end test
    b. Allow it for only super admins

References

- https://www.baeldung.com/spring-boot-security-autoconfiguration
- https://docs.spring.io/spring-security/site/docs/4.2.12.RELEASE/apidocs/org/springframework/security/crypto/password/NoOpPasswordEncoder.html
- https://www.baeldung.com/role-and-privilege-for-spring-security-registration
- https://www.baeldung.com/spring-security-method-security
- https://docs.spring.io/spring-security/site/docs/3.0.x/reference/el-access.html
- https://www.baeldung.com/spring-expression-language