Spring Security in Spring Boot 3

자바에 기반한 앱을 보호하는

프레임워크

1. Introduction

- Spring Security is a framework for securing Java-based applications with great flexibility and customizability

  유연성과 커스터마이징 우수

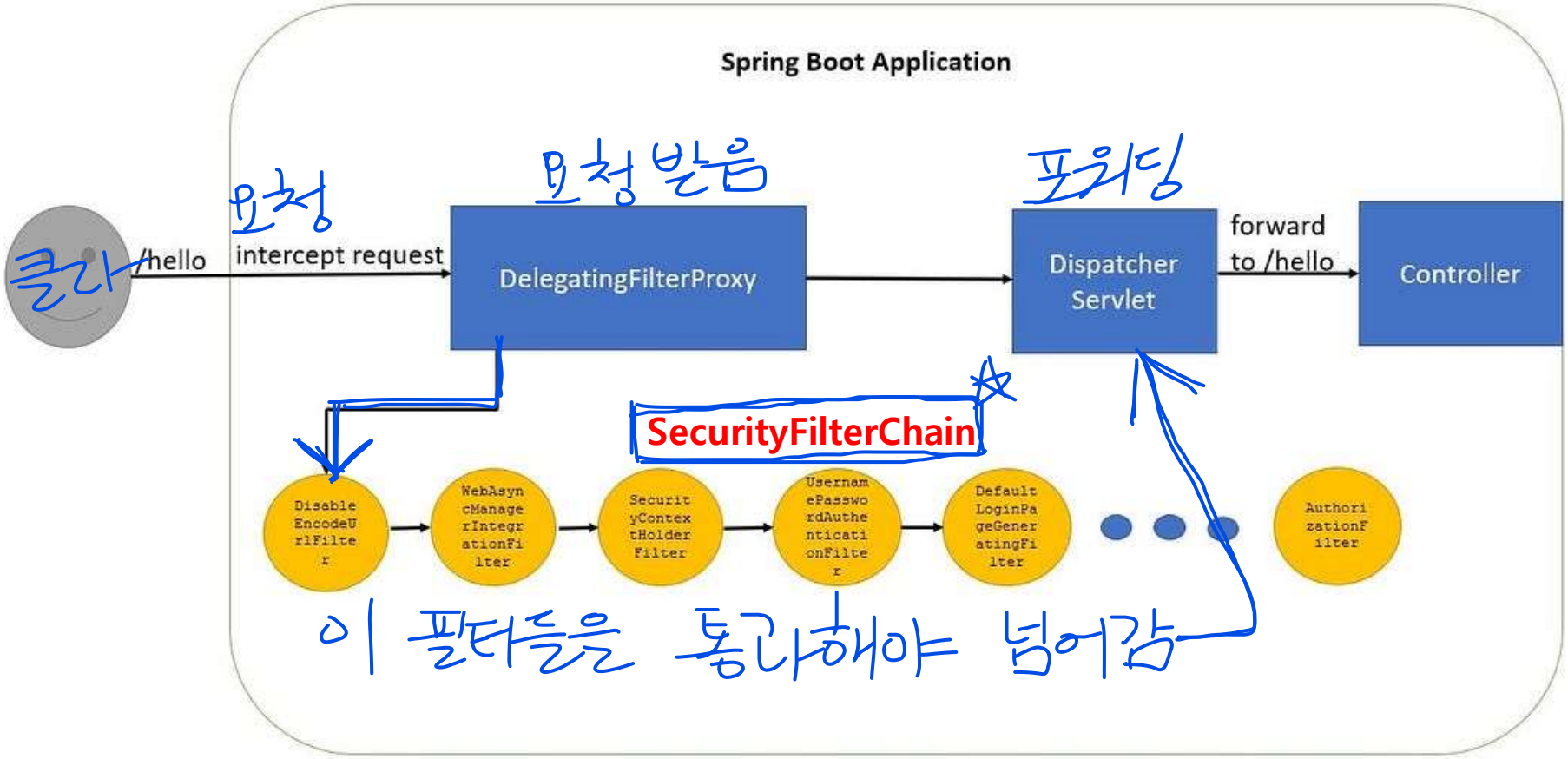
- Spring Security provides authentication and authorization support

  인증과 접근제어 기능을 제공

- Spring Boot provides a **spring-boot-starter-security** starter that aggregates Spring Security-related dependencies together

  를 추가하면 사용 가능

시큐리티의 구조
Introduction



자동 설정

Spring Boot Auto Configuration

- Enables Spring Security's default configuration
- Creates default user with a username as **user** and a randomly generated password that is logged to the console(Ex: 8e557245-73e2-4286-969a-ff57fe326336). user 가 생성됨
- Spring boot provides properties to customize default user's username and password Bcrypt 패스워드 암호화
- Protects the password storage with **Bcrypt** algorithm
- Lets the user log out (default logout feature)
- CSRF attack prevention (enabled by default)
- If Spring Security is on the classpath, Spring Boot automatically secures all HTTP endpoints with "basic" authentication

CSRF가 활성화되어 있으면, 클라는 요청할 때 CSRF 토큰 보내야함

2. Set up Spring Security

```
<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>

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

시큐리티 사용하려면 추가

Set up Spring Security

관련 라이브러리들과 자동설정이 추가된다

- "Spring-boot-starter-security" aggregates spring security-related dependencies and provides auto-configuration

  - Register the AuthenticationManager bean with an in-memory store and a single user (default username: **user**)

```
Using generated security password: 3e7b1c2a-279b-41f2-9775-73aeb5adce62
```

  - Ignore paths for commonly used static resource locations (such as /css/**, /js/**, /images/**, etc.)

  - Enable common low-level features such as XSS, CSRF, caching, etc.

Set up Spring Security

defaut user 정보

- You can change the default user credentials in application.properties

DB가 아닌 메모리에 저장되는 임시 정보임

```
spring.security.user.name=alice
spring.security.user.password=alicepw
spring.security.user.roles=USER, ADMIN
```

4. Configure Spring MVC

- You need to configure Spring MVC and set up view controllers to expose these templates

```
@Configuration
public class WebConfig implements WebMvcConfigurer
{
    @Override
    public void addViewControllers(ViewControllerRegistry registry)
    {
        registry.addViewController("/").setViewName("home");
    }

    @Bean
    public SpringSecurityDialect securityDialect() {
        return new SpringSecurityDialect();
    }
}
```

dependency 이 있음

이것을 한 줄로 표현

```
@Controller
public class HomeController {

    @GetMapping("/")
    public ModelAndView home() {
        return new ModelAndView("home");
    }
}
```

5. Spring Security with Thymeleaf

- "Spring Security Dialect" is a Thymeleaf extras module which helps integrate both of these together(Spring Security + Thymeleaf)

```
<dependency>
    <groupId>org.thymeleaf.extras</groupId>
    <artifactId>thymeleaf-extras-springsecurity6</artifactId>
</dependency>
```

Spring Security with Thymeleaf

- The Spring Security dialect allows us to conditionally display content based on user roles, permissions or other security expressions
- It also gives us access to the Spring *Authentication* object

Thymeleaf 로도 시큐리티 사용 가능

```
<!DOCTYPE html>
<html   xmlns:th=http://www.thymeleaf.org
        xmlns:sec="http://www.thymeleaf.org/thymeleaf-extras-springsecurity6">
    <head>
        <title>Welcome to Spring Security tutorial</title>
    </head>
    <body>
        <h2>Welcome</h2>
        <p>Spring Security tutorial</p>
        <div sec:authorize="hasRole('USER')">Text visible to user.</div>
        <div sec:authorize="hasRole('ADMIN')">Text visible to admin.</div>
        <div sec:authorize="isAuthenticated()"> Text visible only to authenticated users. </div>

          Authenticated username:
        <div sec:authentication="name"></div>

    </body>
</html>
```

시큐리티 설정 ☆

6. Configure Spring Security

DB에 사용자 정보 만들기

1) Role-based access control using a database
   - User JPA Entity, Role JPA Entity (N:N mapping) ← N:N 매핑
   - Spring Data JPA repository for the User, Role entities

user에 대한 JPA Entity 와 role에 대한 JPA Entity 만든다

2) UserDetailsService to get UserDetails from database
   - loadUserByUsername ( )

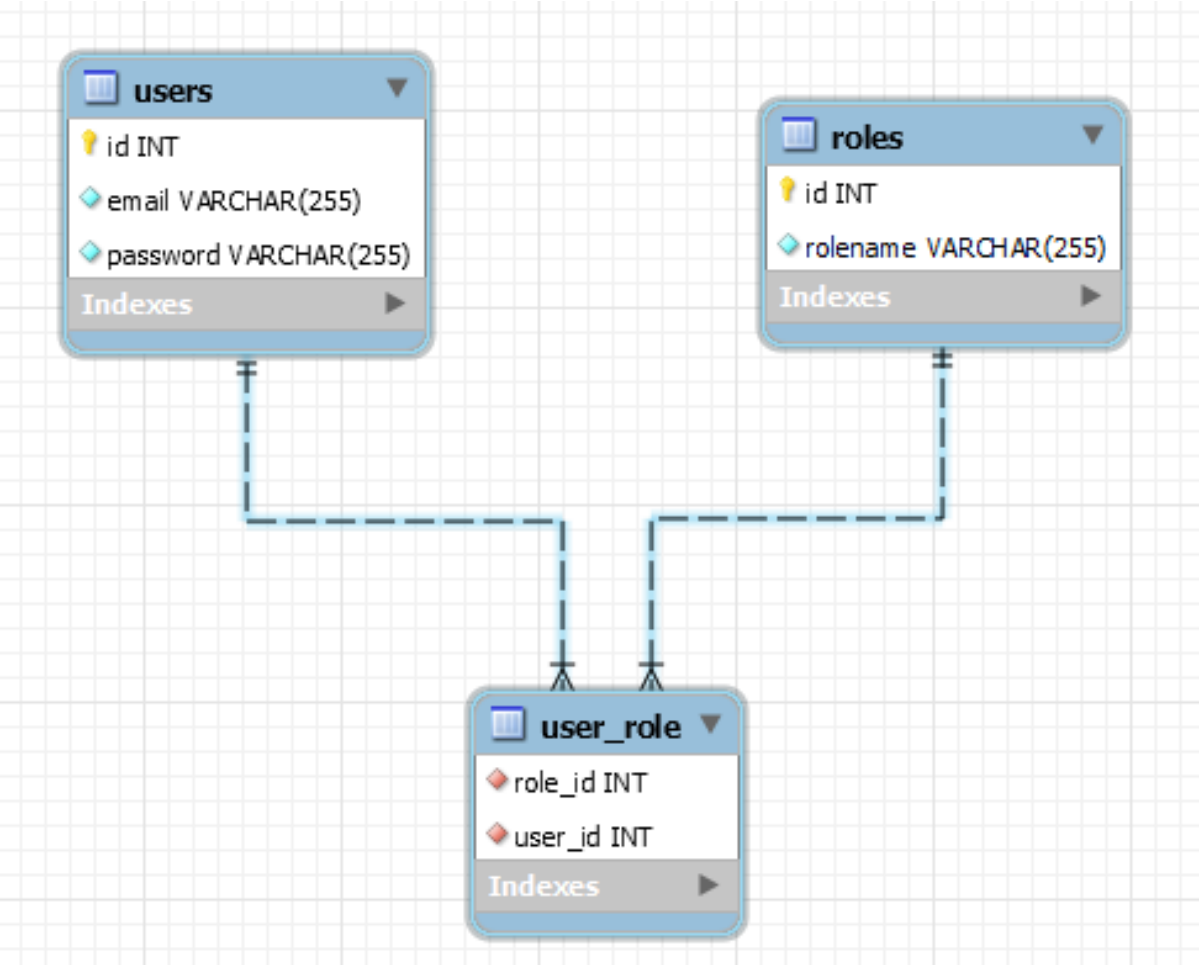
UserDetailsService 라는 인터페이스에 있는 함수 구현해줘야함

user랑 role을 스프링은 모른다 →

3) Customized Spring Security Configuration
   - Authentication, Authorization

인증과 접근제어도 설정해줘야함

1) Create user and role entities



Create user and role entities

pom.xml

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

<dependency>
        <groupId>com.mysql</groupId>
        <artifactId>mysql-connector-j</artifactId>
</dependency>
```

application.properties

```
# DataSource Setting
spring.datasource.url=jdbc:mysql://localhost:3306/member?useSSL=false&characterEncoding=UTF-8&serverTimezone=Asia/Seoul
spring.datasource.username=root
spring.datasource.password=csedbadmin
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.sql.init.mode=always
spring.sql.init.encoding= UTF-8

# JPA Setting
spring.jpa.hibernate.ddl-auto=create
spring.jpa.show-sql=true
spring.jpa.properties.hibernate.format_sql=false

# After the ddl-auto execution, data.sql is executed and the data is applied
spring.jpa.defer-datasource-initialization=true

# Logging Level Setting
logging.level.kr.ac.hansung=debug
```

You need to execute the following sql statement

create database member default character set utf8 collate utf8_general_ci;

User.java

```
@Entity
@Table(name="users")
@Getter
@Setter
@NoArgsConstructor
public class User
{
    @Id
    @GeneratedValue(strategy= GenerationType.IDENTITY)
    private Integer id;

    @Column(nullable=false)
    private String password;

    @Column(nullable=false, unique=true)
    private String email;

    @ManyToMany(cascade=CascadeType.MERGE) @JoinTable(
            name="user_role",
            joinColumns={@JoinColumn(name="USER_ID", referencedColumnName="ID")},
            inverseJoinColumns={@JoinColumn(name="ROLE_ID", referencedColumnName="ID")})
    private List<Role> roles;
}
```

Role.java

```
@Entity
@Table(name="roles")
@Getter
@Setter
public class Role
{
    @Id
    @GeneratedValue(strategy= GenerationType.IDENTITY)
    private Integer id;

    @Column(nullable=false, unique=true)
    private String rolename;

    @ManyToMany(mappedBy="roles")
    private List<User> users;

    public Role(String rolename) {
        this.rolename = rolename;
    }
}
```

UserRepository.java

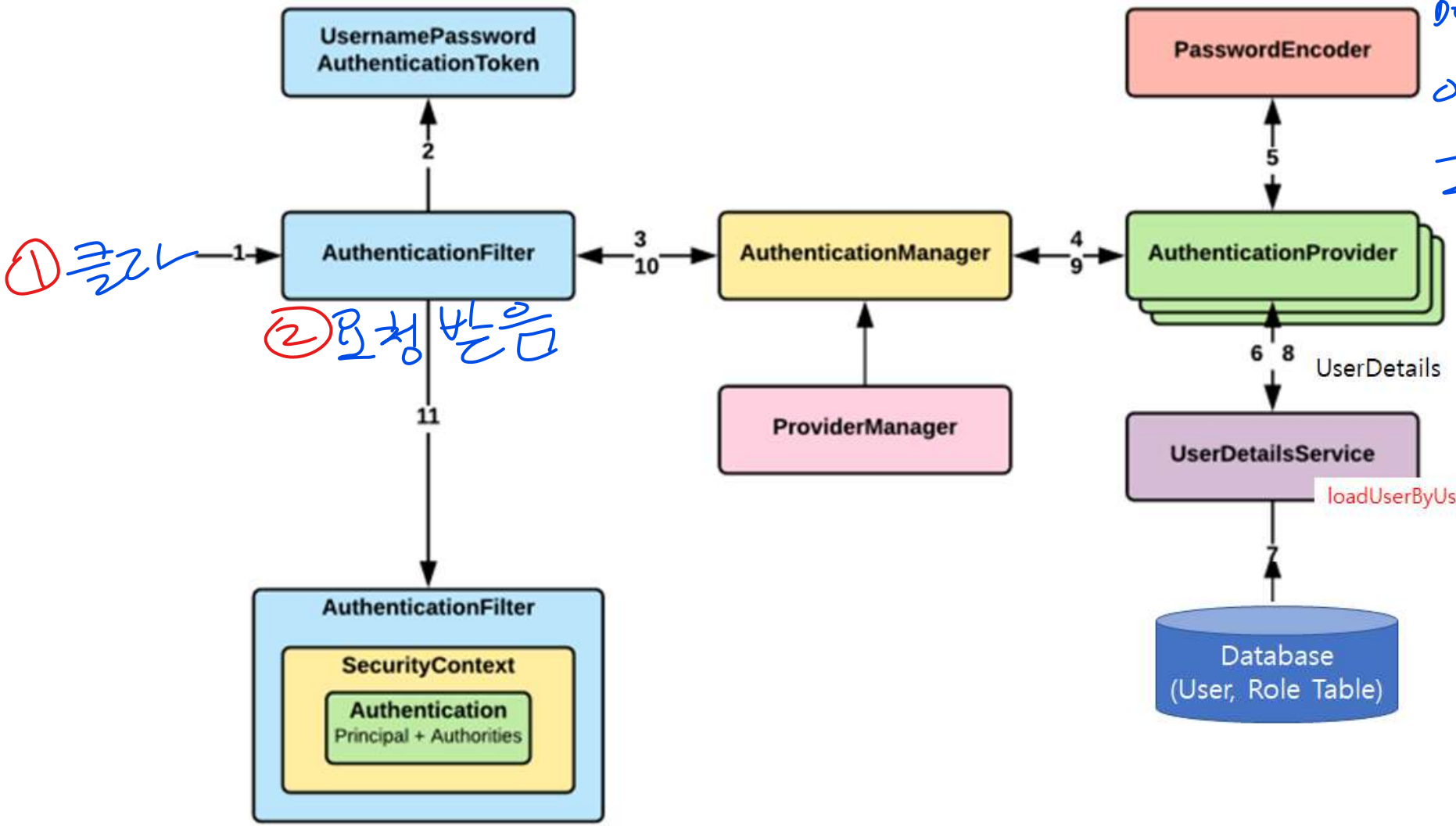
```
public interface UserRepository extends JpaRepository<User, Integer>
{
    Optional<User> findByEmail(String email);
}
```

null exception 예외를 처리하기 위해 씀

RoleRepository.java

```
public interface RoleRepository extends JpaRepository<Role, Integer> {
    Optional<Role> findByRolename(String rolename);
}
```

2) UserDetailsService Interface



① 클라 ② 요청 받음

③ 이것은 컨테이너의 bean으로 등록해주고 이것은 비밀번호를 확인. username과 password가 오면

④ 그리고 UserDetails를 리턴

⑤ 함수를 호출

UserDetailsService Interface

- The *AuthenticationProvider* uses *UserDetailsService* interface to load details about the user during authentication

- The *UserDetailsService* interface has one method named *loadUserByUsername()* which can be overridden to customize the process of finding the user

이 함수는 username을 받아서 객체를 리턴한다

UserDetailsService Interface

- The *loadUserByUsername* method returns a *UserDetail* object, which is also an interface and contains some methods for describing user information

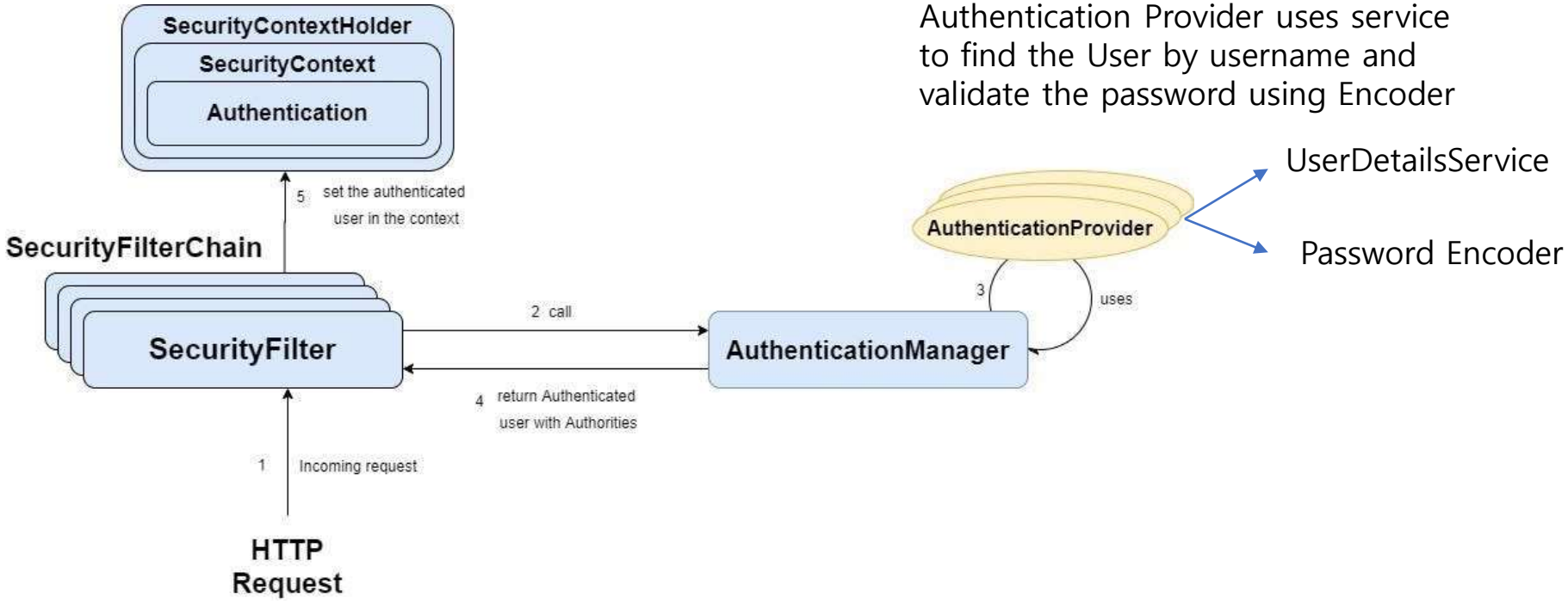
- Spring Security provides an out-of-the box implementation of org.springframework.security.core.userdetails.User

UserDetailsService Interface

```
@Service
@Transactional
public class CustomUserDetailsService implements UserDetailsService
{
    @Autowired
    private UserRepository userRepository;

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

        User user = userRepository.findByEmail(userName)
                .orElseThrow(() -> new UsernameNotFoundException("Email: " + userName + " not found"));

        return new org.springframework.security.core.userdetails.User(user.getEmail(),
                user.getPassword(), getAuthorities(user));
    }
    ...
}
```

Optional<User>

3) Spring Security Configuration



Authentication Provider uses service to find the User by username and validate the password using Encoder

SecurityFilterChain

- Replaces the older *WebSecurityConfigurerAdapter* in Spring Boot 3 for configuring HTTP security
- A collection of security filters, ranging from Security Filter_0 to Security Filter_n
- Each filter performs specific security tasks such as authentication, authorization, and CSRF protection
- Configures the 'SecurityFilterChain' using the `HttpSecurity` object

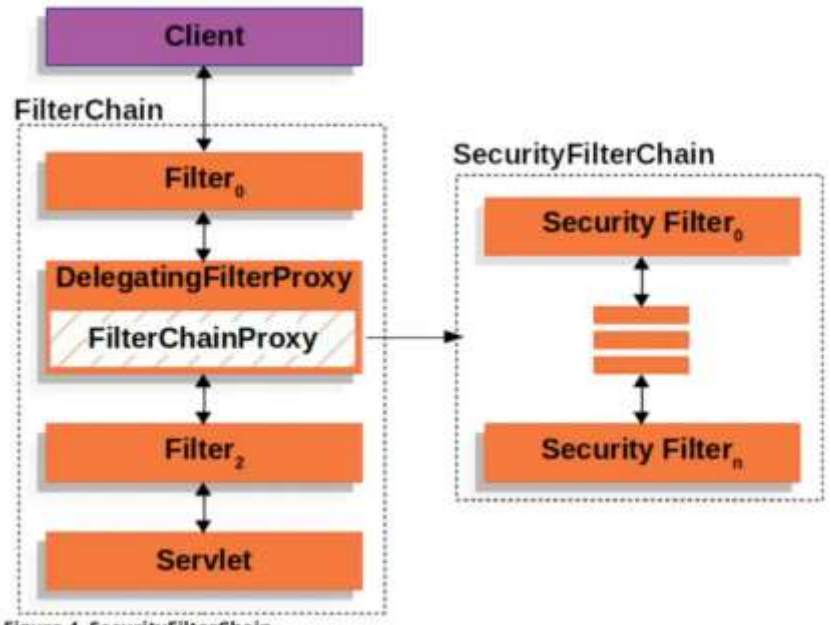


Figure 4. SecurityFilterChain

Spring Security Configuration
(sample code)

```
@Configuration
@EnableWebSecurity
public class WebSecurityConfig {

    @Autowired
    private UserDetailsService customUserDetailsService;

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

    private static final String[] PUBLIC_MATCHERS = {
            "/css/**",
            "/js/**",
            "/images/**"
    };
```

```
@Bean
    public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
        http
                .authorizeHttpRequests(authz -> authz
                        .requestMatchers(PUBLIC_MATCHERS).permitAll()
                        .requestMatchers("/", "/home", "/signup").permitAll()
                        .requestMatchers("/admin/**").hasRole("ADMIN")
                        .anyRequest().authenticated()
                )
                .formLogin(formLogin -> formLogin
                        .loginPage("/login")
                        .defaultSuccessUrl("/home")
                        .failureUrl("/login?error")
                        .permitAll()
                )
                .logout(logout -> logout
                        .logoutUrl("/logout")
                        .logoutSuccessUrl("/login?logout")
                        .permitAll()
                )
                .userDetailsService(customUserDetailsService)
                .csrf(AbstractHttpConfigurer::disable);

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

SecurityFilterChain을 리턴하여 필터를 설정☆

Spring Security Configuration

- To provide your own customized security configuration, you can create a configuration class that uses *SecurityFilterChain* for defining security rules
  - authorizeHttpRequests: used to configure access control for HTTP requests
  - formLogin and logout: configured to handle login and logout processes, specifying URLs for redirection on success and failure

- This example configures CustomUserDetailsService and BCryptPasswordEncoder to be used by AuthenticationManager

4) BCryptPasswordEncoder

Bcrypt will internally generate **a random salt**.
This is important to understand because it means that each call will have a different result

$2a$10$N9qo8uLOickgx2ZMRZoMyeIjZAgcfl7p92ldGxad68LJZdL17lhWy

항상 비밀번호는 같지만, 이러한 salt 값을 붙여 구분하는 것임,

There are three fields separated by $:
The *"2a"* : the BCrypt algorithm version
The *"10"* : the strength of the algorithm
The "N9qo8uLOickgx2ZMRZoMye" : the randomly generated salt
16-byte (128-bit), base64-encoded to 22 characters
The remaining part : the actual hashed version of the plain text
24-byte (192-bit) hash, base64-encoded to 31 characters

salt 값은 공개가 되도 문제없다

BCryptPasswordEncoder

```
@SpringBootTest
class HellospringsecurityApplicationTests {

    @Autowired
    private PasswordEncoder encoder;

    @Test
    void generateHashedPassword() {
        String pwd = encoder.encode("alicepw");
        System.out.println(pwd);
    }

}
```

$2a$10$yiTi0281XvoHr9aEVl.r5.U.vOg1nSFUCVxg9hn3ZjZH/I86hRstu

7. Create views using Thymeleaf

- home.html, login.html, adminhome.html, 403.html

login.html

```
<form action="login" th:action="@{/login}" method="post">
    <div th:if="${param.error}">
        <span style="color:red">Invalid Email and Password.</span>
    </div>
    <div th:if="${param.logout}">
        <span style="color:blue">Successfully logged out</span>
    </div>

    <label for="id-email">Email:</label><br>
    <input type="email" id="id-email" name="username" placeholder="Email"><br>
    <label for="id-password">Password:</label><br>
    <input type="password" id="id-password" name="password" placeholder="Password" />

    <button type="submit">LogIn</button>
</form>
```

login failureUrl : "/login?error"
logoutSuccessUrl : "/login?logout"

Create views using Thymeleaf

Modify Configuration class for providing MVC configuration

```
@Configuration
public class WebConfig implements WebMvcConfigurer
{
    @Override
    public void addViewControllers(ViewControllerRegistry registry)
    {
        registry.addViewController("/login").setViewName("login");
        registry.addViewController("/home").setViewName("home");
        registry.addViewController("/admin/home").setViewName("adminhome");
        registry.addViewController("/accessDenied").setViewName("403");
    }
}
```

8. Cross-Site Request Forgery

- If you are using Spring Security and Thymeleaf, the CSRF token will be automatically included if the <form> has the th:action attribute and method is anything other than GET, HEAD, TRACE, or OPTIONS

```
<form th:action="@{/messages}" method="post">
        <textarea name="content" cols="50" rows="5"></textarea>
        <input type="submit" value="Submit"/>
</form>
```

- When it is rendered, if you see the page source, you can see the CSRF token inserted automatically as a hidden parameter

```
<form action="/messages" method="post">
        <input type="hidden" name="_csrf" value="57f12f98-d62c-4d01-88c0-a11cf9ed980a"/>
        <textarea name="content" cols="50" rows="5"></textarea>
        <input type="submit" value="Submit"/>
 </form>
```

9. SignUp Page

- RegistrationService Interface
- RegistrationServiceImpl Class

- RegistrationController Class

- Siginup.html

Thymeleaf View(Signup Form)

```
<div align="center">
    <form  action="#" th:action="@{/signup}" th:object="${user}" method="post">
        <table border="0" cellpadding="10">
            <tr>
                <div th:if="${emailExists}">
                    <span style="color:red">Email already exists</span>
                </div>
                <td>Your Email:</td>
                <td><input type="text" th:field="*{email}" /></td>
            </tr>
             <tr>
                <td>Password:</td>
                  <td><input type="password" th:field="*{password}" /></td>
            </tr>

            <tr>
                <td colspan="2"><button type="submit">SignUp</button>  </td>
            </tr>
        </table>
    </form>
</div>
```