
spring

JPA 블로그 프로젝트 실습

- 엔티티에 생성, 수정 시간 추가하기

```java
package com.kosta.entity;

@Entity // 엔티티 지정
@EntityListeners(AuditingEntityListener.class)
// 엔티티의 생성 및 수정 시간 자동으로 감시
@RequiredArgsConstructor
@Data
public class Article {
    // 중략…
    @CreatedDate
    @Column(name="created_at")
    private LocalDateTime createdAt;

    @LastModifiedDate
    @Column(name="updated_at")
    private LocalDateTime updatedAt;
}
```

JPA 블로그 프로젝트 실습

- 작성일, 수정일 자동 업데이트 위한 설정

```
package com.kosta;

@EnableJpaAuditing // created_at, updated_at 자동으로 업데이트
@SpringBootApplication
public class BlogProjApplication {
    public static void main(String[] args) {
        SpringApplication.run(BlogProjApplication.class, args);
    }
}
```

JPA 블로그 프로젝트 실습

- /blog/* 로 동작되도록 경로 변경 (HTML 파일 모두)

```
@Controller
@RequestMapping("/blog/*")
@RequiredArgsConstructor
public class BlogController {
    // 생략
}
```

```
return "redirect:/blog/list";
```

Spring Security

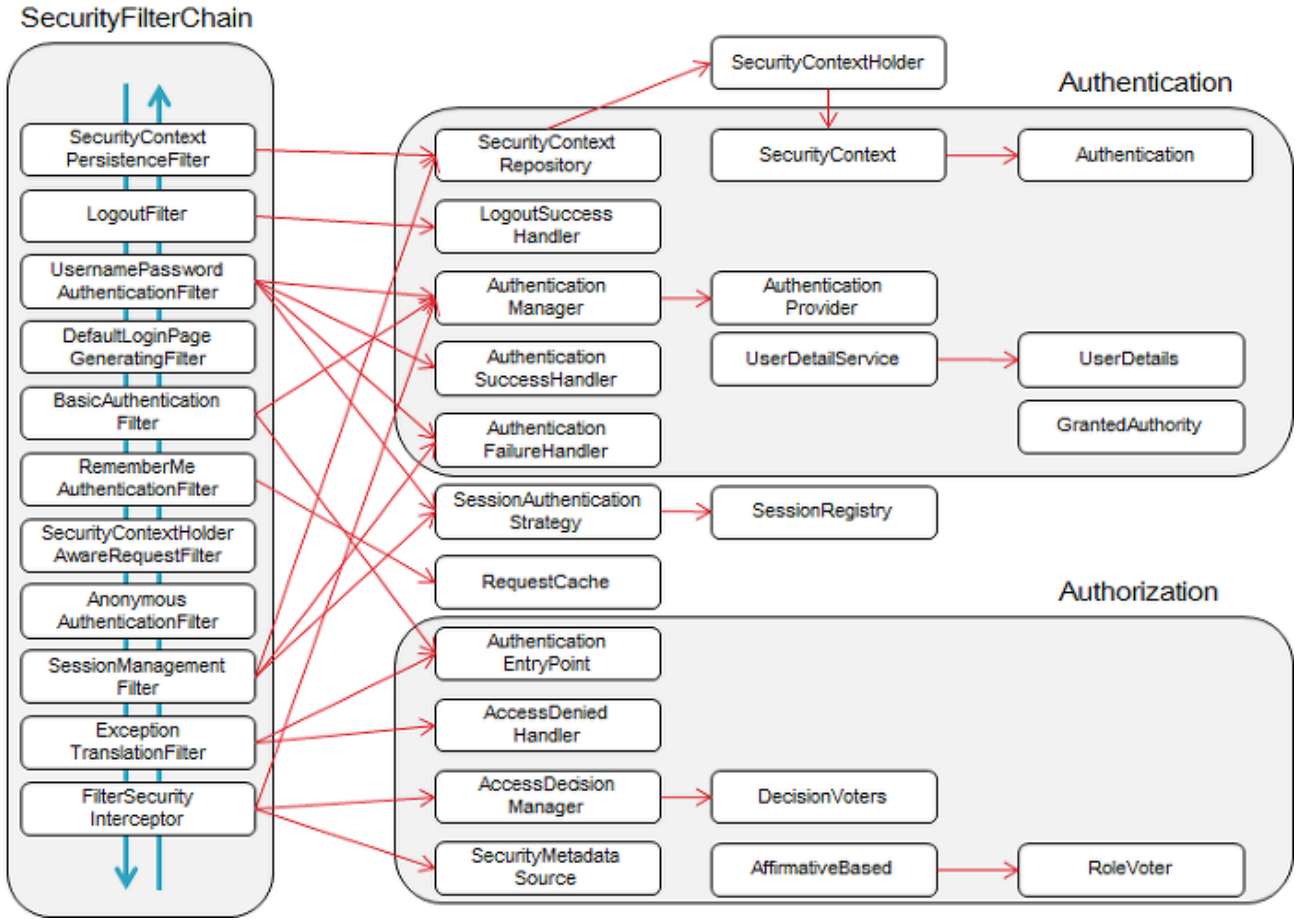
- 스프링 시큐리티는 스프링 기반의 웹 애플리케이션에서 보안(인증, 인가, 권한 등)을 담당하는 프레임워크이다.
- 스프링 시큐리티를 이해하기 위해서는 인증과 인가에 대한 개념을 알아야 한다.

- 인증(Authentication)은 사용자가 누구인지 확인하는 과정을 의미한다.
- 인가(Authorization)은 특정 부분에 접근할 수 있는 권한을 확인하는 과정을 의미한다.

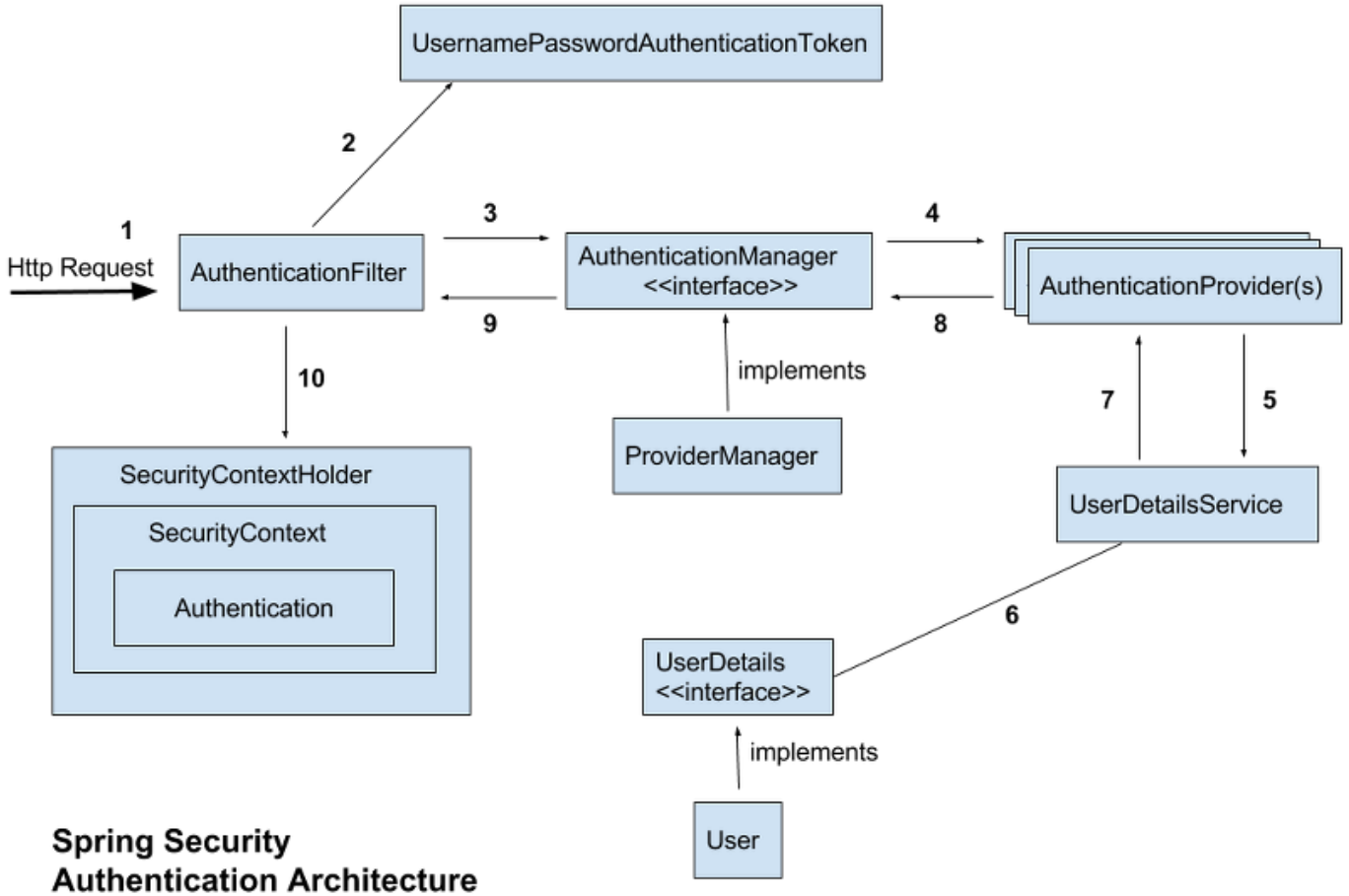
- 인증과 인가 관련 코드를 작성하려면 굉장히 많은 시간이 필요하지만, 스프링 시큐리티를 사용하면 아주 쉽게 처리할 수 있다.

Spring Security

- 스프링 시큐리티는 필터를 기반으로 동작한다.
- 스프링 시큐리티에는 다양한 필터들이 있으며, 각 필터에서 인증, 인가와 관련된 작업을 처리하게 된다.



Spring Security



Spring Security

- 먼저 스프링 시큐리티를 사용하기 위해 build.gradle 파일에 의존성을 추가하고 refresh 한다.

```
// 스프링 시큐리티를 사용
implementation 'org.springframework.boot:spring-boot-starter-security'
// 타임리프에서 스프링 시큐리티를 사용
implementation 'org.thymeleaf.extras:thymeleaf-extras-springsecurity6'
// 스프링 시큐리티 테스트 사용
testImplementation 'org.springframework.security:spring-security-test'
```

Spring Security

- 회원 엔티티와 매핑할 테이블 구조를 확인하자

컬럼명	자료형	null 허용	키	설명
id	BIGINT	X	기본키	회원 일련변호
email	VARCHAR(255)	X		이메일
password	VARCHAR(255)	X		패스워드(암호화)
created_at	DATETIME	X		생성일자
updated_at	DATETIME	X		수정일자

Spring Security

- 회원 엔티티를 생성하자

```java
package com.kosta.entity;

@Entity
@EntityListeners(AuditingEntityListener.class) // 엔티티의 생성 및 수정 시간 자동으로 감시
@RequiredArgsConstructor
@Data
public class User implements UserDetails {
	@Id // 기본키 지정
	@GeneratedValue(strategy = GenerationType.IDENTITY) // 자동 증가
	@Column(name="id", updatable = false) // update 시에 컬럼에 포함하지 않음
	private Long id;

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

	@Column(name="password", nullable = false)
	private String password;

	@CreatedDate
	@Column(name="created_at")
	private LocalDateTime createdAt;

	@LastModifiedDate
	@Column(name="updated_at")
	private LocalDateTime updatedAt;


	@Builder // 빌더 패턴으로 객체 생성
	public User(String email, String password, String auth) {
		this.email = email;
		this.password = password;
	}
}
```

Spring Security

- 빌더 패턴를 생성하고, UserDetail 인터페이스의 메소드를 오버라이드하자. (getPassword()는 Getter로 자동 오버라이딩 된다.)

```
@Builder // 빌더 패턴으로 객체 생성
public User(String email, String password, String auth) {
      this.email = email;
      this.password = password;
}

@Override // 사용자가 가진 권한 목록 반환 (사용자 권한만 있기 때문에 "user"만 반환)
public Collection<? extends GrantedAuthority> getAuthorities() {
      return List.of(new SimpleGrantedAuthority("user"));
}

@Override // 사용자 식별 가능한 이름 반환
public String getUsername() { return email; }

@Override // 계정 만료 여부
public boolean isAccountNonExpired() { return true; }

@Override // 계정 잠금 여부
public boolean isAccountNonLocked() { return true; }

@Override // 비밀번호 만료 여부
public boolean isCredentialsNonExpired() { return true; }

@Override // 계정 사용 가능 여부
public boolean isEnabled() { return true; }
```

Spring Security

- UserRepository 인터페이스 생성 및 findByEmail() 메소드 작성

```
package com.kosta.repository;

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

Spring Security

- 로그인을 진행할 때 사용자 정보를 가져오는 UserDetailsServiceImpl 생성

```
package com.kosta.service;

@Service
@RequiredArgsConstructor
public class UserDetailsServiceImpl implements UserDetailsService {
    private final UserRepository userRepository;

    @Override
    public User loadUserByUsername(String email) {
        User user = userRepository.findByEmail(email)
            .orElseThrow(() -> new IllegalArgumentException(email));
        return user;
    }
}
```

Spring Security

- 스프링 시큐리티 설정을 위해 com.kosta.config.WebSecurityConfig 작성

```
@Configuration
@EnableWebSecurity
@RequiredArgsConstructor
public class WebSecurityConfig {

    private final UserDetailsService userService;

    // 특정 부분에 스프링 시큐리티 기능 비활성화
    @Bean
    WebSecurityCustomizer configure() {
        return (web) -> web.ignoring().requestMatchers(new AntPathRequestMatcher("/static/**"));
    }


}
```

Spring Security

- 스프링 시큐리티 설정을 위해 com.kosta.config.WebSecurityConfig 작성

```
@Configuration
@EnableWebSecurity
@RequiredArgsConstructor
public class WebSecurityConfig {

	// 생략

	// 특정 HTTP 요청에 대한 보안 구성
	@Bean
	SecurityFilterChain filiterChain(HttpSecurity http) throws Exception {
		return http
		.authorizeHttpRequests(auth -> auth.requestMatchers( // 인증, 인가 설정 (특정 요청 URL 액세스 설정, 나머지는 인증 필요)
			new AntPathRequestMatcher("/login"),
			new AntPathRequestMatcher("/join")
		).permitAll().anyRequest().authenticated())
		.formLogin(formLogin -> formLogin.loginPage("/login").defaultSuccessUrl("/blog/list")) // 폼 기반 로그인 설정
		.logout(logout -> logout.logoutSuccessUrl("/login").invalidateHttpSession(true)) // 로그아웃 설정
		.cors(AbstractHttpConfigurer::disable) // CORS 비활성화
		.csrf(AbstractHttpConfigurer::disable) // CSRF 비활성화 (공격 방지)
		.build();
	}
}
```

Spring Security

- 스프링 시큐리티 설정을 위해 com.kosta.config.WebSecurityConfig 작성

```
@Configuration
@EnableWebSecurity
@RequiredArgsConstructor
public class WebSecurityConfig {

	// 생략

	// 인증 관리자 관련 설정
	@Bean
	AuthenticationManager authenticationManager(
			HttpSecurity http, BCryptPasswordEncoder bCryptPasswordEncoder,
			UserDetailsService userDetailsService) throws Exception {
		DaoAuthenticationProvider authProvider = new DaoAuthenticationProvider();
		authProvider.setUserDetailsService(userService);
		authProvider.setPasswordEncoder(bCryptPasswordEncoder);
		return new ProviderManager(authProvider);
	}

	// 비밀번호 암호화 사용 설정
	@Bean
	BCryptPasswordEncoder bCryptPasswordEncoder() {
		return new BCryptPasswordEncoder();
	}
}
```

Spring Security

- 회원 관련 서비스 UserService, UserServiceImpl 생성

```
package com.kosta.service;

public interface UserService {
     Long Save(User user);
}
```

```
package com.kosta.service;

@Service
@RequiredArgsConstructor
public class UserServiceImpl implements UserService {
     private final UserRepository userRepository;
     private final BCryptPasswordEncoder bCryptPasswordEncoder;

     @Override
     public Long Save(User user) {
          user.setPassword(bCryptPasswordEncoder.encode(user.getPassword()));
          return userRepository.save(user).getId();
     }
}
```

Spring Security

- 회원 컨트롤러 작성

```
@Controller
@RequiredArgsConstructor
public class UserController {
	private final UserService userService;

	@GetMapping("/login")
	public String loginPage() {
		return "login";
	}

	@GetMapping("/join")
	public String joinPage() {
		return "join";
	}

	@GetMapping("/logout")
	public String logout(HttpServletRequest req, HttpServletResponse res) {
		new SecurityContextLogoutHandler().logout(req, res, SecurityContextHolder.getContext().getAuthentication());
		return "redirect:/login";
	}

	@PostMapping("/join")
	public String join(User user) {
		userService.Save(user);
		return "redirect:/login";
	}
}
```

Spring Security

- 로그아웃버튼추가

```
<div class="p-5 mb-5 text-center bg-dark-subtle">
    <h1 class="mb-3">My Blog</h1>
    <h4 class="mb-3">환영합니다.</h4>
    <button class="btn btn-secondary btn-sm" onclick="location.href='/logout'">로그아웃</button>
</div>
```

Spring Security

- 회원가입, 로그인 뷰 작성