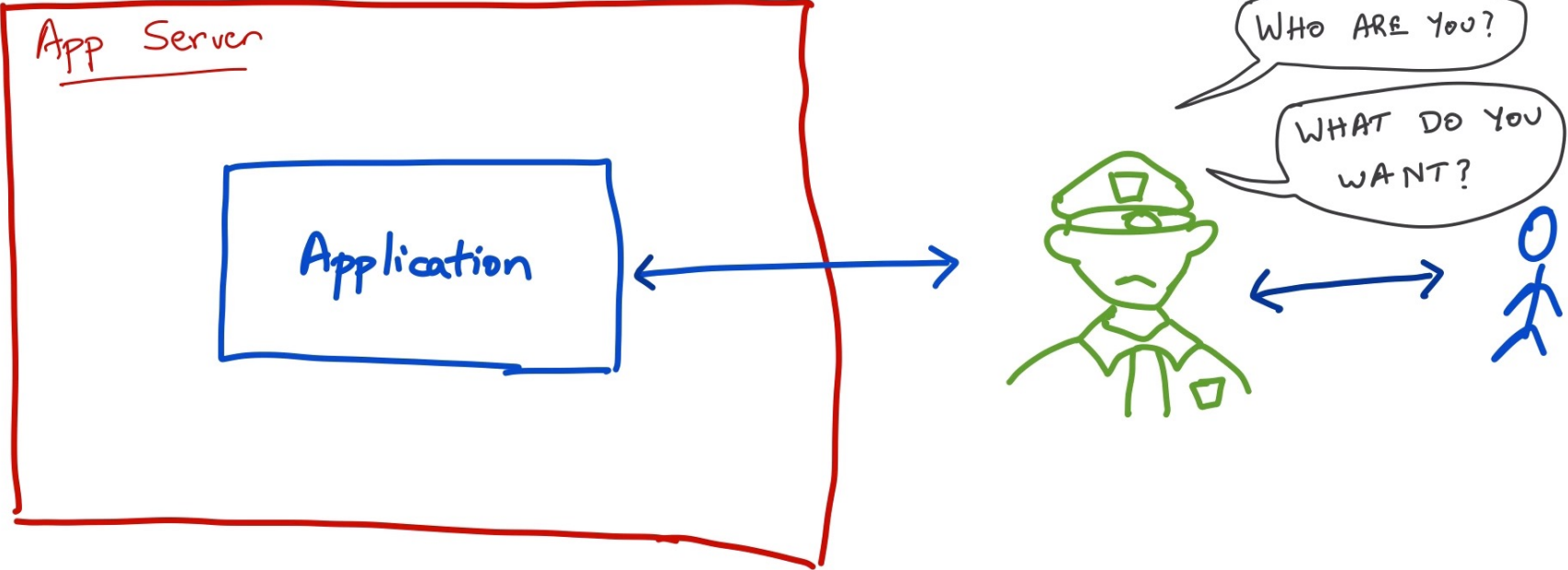


Spring Security

Uygulama güvenliği nedir?

- ▶ Pek çok farklı güvenlik türü vardır: İşletim sistemi güvenliği, ağ güvenliği, dosya güvenliği, veri tabanı güvenliği, uygulama güvenliği vs.
- ▶ Uygulama güvenliği, web uygulamamızın güvenli hale getirilmesidir.
- ▶ Ana olarak iki soru ile ilgilenir:
 - ▶ İsteği yapan kişi kim?
 - ▶ Benden ne istiyor ve bunu yapmaya yetkisi var mı?



Neden Spring Security?

- ▶ Kullanmaya başladığınız anda clickjacking, CSRF, session fixation gibi yaygın saldırılara karşı koruma sağlaması
- ▶ Son derece esnek ve açık bir yapı sağlaması
- ▶ Endüstride çok yaygın bir şekilde kullanılması
- ▶ Öğrenme eğrisi keskin olmasına rağmen öğrendikten sonra çok güçlü özelliklerinin bulunması

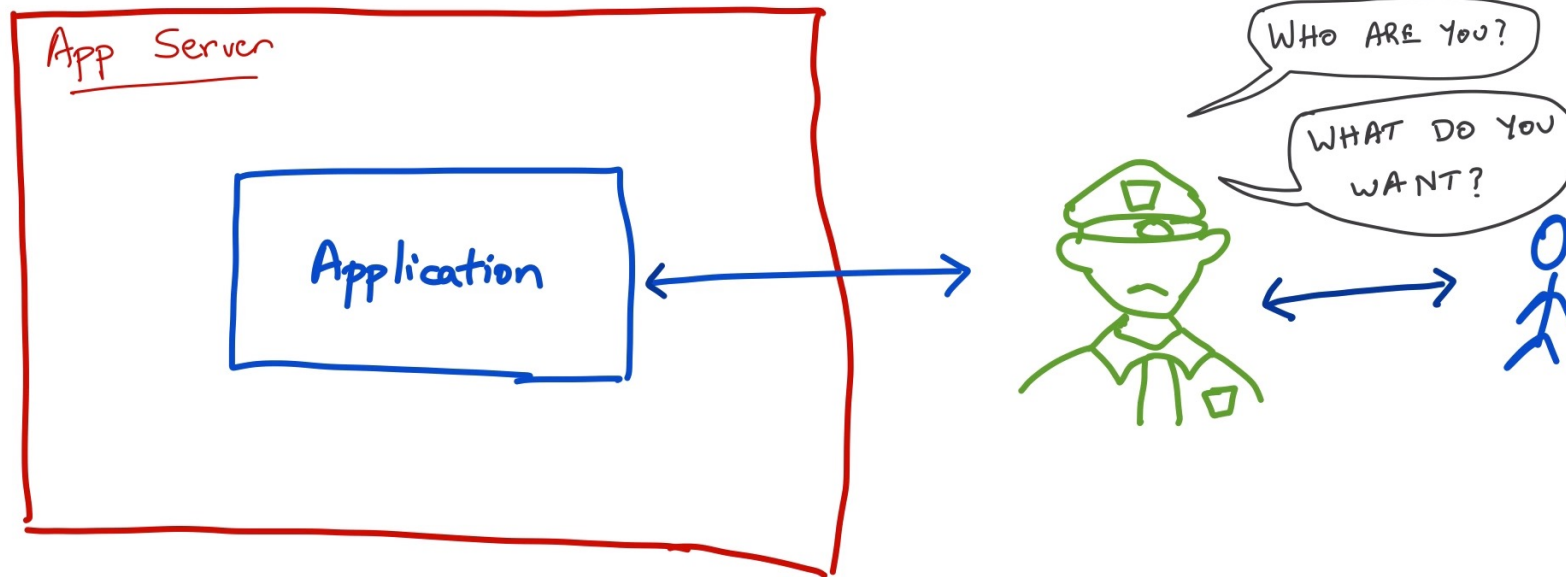
5 temel konsept

- ▶ Authentication
- ▶ Authorization
- ▶ Principal
- ▶ Granted Authority
- ▶ Roles

Authentication ve Authorization

- ▶ Authentication, isteği yapan kişinin kim olduğu ile ilgilenir
- ▶ Authorization, authenticate olan kullanıcının hangi yetkileri olduğuyla ilgilenir

javabrain.io

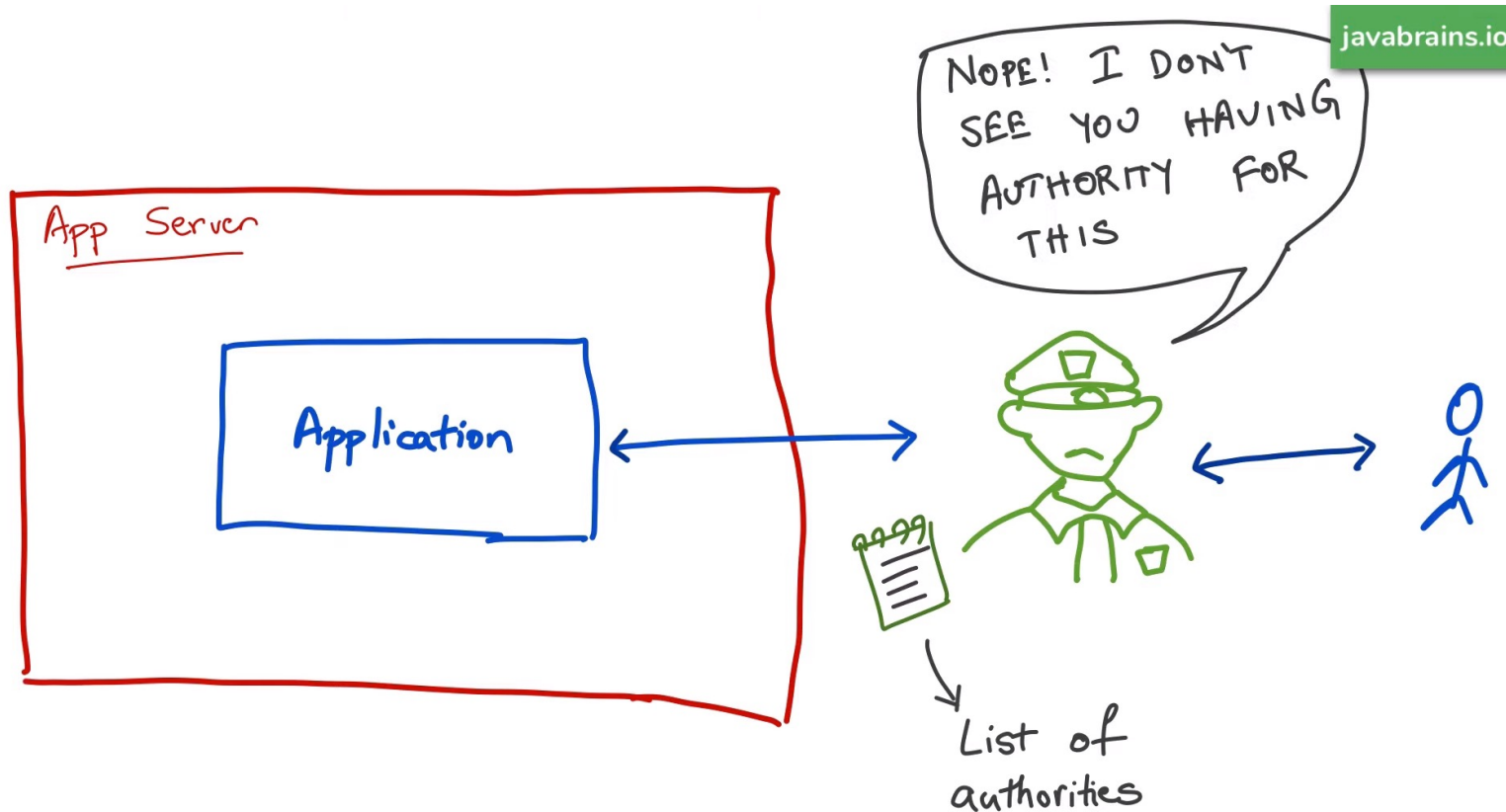


Principal

- ▶ Principal, giriş yapmış olan kullanıcıyı temsil eder.
- ▶ Kullanıcı adı, şifre gibi tanımlayıcı bilgiler içeririr.

Granted Authority

- ▶ Giriş yapmış olan kişinin hangi yetkilere sahip olduğunu gösterir
- ▶ Bir kullanıcının, neler yapabileceğine dair detaylı yetkiler
- ▶ Her kullanıcının, genellikle birden fazla granted authority'si bulunur.



Role

- ▶ Granted authority grupları
- ▶ Öğretmen, öğrenci, admin vs gibi

Spring security dependency ekleme





PRACTiCE

Uygulama

- ▶ Bir RestController ekleyin ve orada kendi isminizi yazan bir string dönün
- ▶ Buraya bir URL atayın ve tarayıcıdan buraya erişmeye çalışın
- ▶ Sizi login ekranına yönlendirecek. Kullanıcı adına user, şifreye konsolda yazan şifreyi girin ve URL'e erişin
- ▶ Browser'dan /logout sayfasına gidip, aynı URL'e tekrar erişmeye çalışın, izin vermediğini görün

Security ile ilgili ayarları yapma

WebSecurityConfigurerAdapter

- ▶ Spring security'deki hemen hemen tüm ayarları yapmak için giriş noktası
- ▶ Pek çok farklı ve kullanımı kolay methoduyla konfigürasyon kısmını kolaylaştırıyor
- ▶ Bu class'ı extend ederek başlanır

```
@Configuration
public class SecurityConfiguration {

    @Autowired
    public SecurityConfiguration(AuthenticationManagerBuilder authenticationManagerBuilder)
        throws Exception {
        UserDetails user = User.builder()
            .username("user")
            .password("user")
            .authorities("ROLE_USER")
            .build();

        authenticationManagerBuilder.inMemoryAuthentication()
            .withUser(user)
            .passwordEncoder(NoOpPasswordEncoder.getInstance());
    }
}
```



PRACTiCE

Uygulama

- ▶ Kendi isminizle ve kendi belirleyeceğiniz bir şifre ile bir kullanıcı belirleyin
- ▶ Daha önce oluşturduğumuz rest servise yeni kullanıcı adı ile erişin

PasswordEncoder

- ▶ AuthenticationManagerBuilder, bir password encoder bekliyor
- ▶ Şifreleri, düz metin olarak kaydetmek çok tehlikeli
- ▶ Spring standardı BCrypt encoder

```
@Configuration
public class SecurityConfiguration {

    @Autowired
    public SecurityConfiguration(AuthenticationManagerBuilder authenticationManagerBuilder)
        throws Exception {
        BCryptPasswordEncoder passwordEncoder = new BCryptPasswordEncoder();
        UserDetails user = User.builder()
            .username("user")
            .password(passwordEncoder.encode("user"))
            .authorities("ROLE_USER")
            .build();

        authenticationManagerBuilder.inMemoryAuthentication()
            .withUser(user)
            .passwordEncoder(passwordEncoder);
    }
}
```

Authorization ayarları

Kullanıcı yetkilerini ayarlama

- ▶ Kullanıcı oluştururken, hangi yetkilerinin olması gerektiğini belirleyebiliyoruz
- ▶ Aynı zamanda rol ataması da yapabiliyoruz

```
@Configuration
public class SecurityConfiguration {

    @Autowired
    public SecurityConfiguration(AuthenticationManagerBuilder authenticationManagerBuilder)
        throws Exception {
        BCryptPasswordEncoder passwordEncoder = new BCryptPasswordEncoder();
        UserDetails user = User.builder()
            .username("user")
            .password(passwordEncoder.encode("user"))
            .authorities("READ", "ROLE_USER")
            .build();

        authenticationManagerBuilder.inMemoryAuthentication()
            .withUser(user)
            .passwordEncoder(passwordEncoder);
    }
}
```

URL tabanlı access control

- ▶ Spring Security, hangi URL'in hangi yetkiler veya rollere sahip kişiler tarafından erişilebileceğini ayarlamamızı sağlıyor
- ▶ Aşağıdaki methodlarla yüksek hassasiyet ile ayarlama yapmamızı sağlıyor
 - ▶ permitAll
 - ▶ denyAll
 - ▶ authenticated
 - ▶ hasIpAddress
 - ▶ hasRole
 - ▶ hasAnyRole
 - ▶ hasAuthority
 - ▶ hasAnyAuthority

@Configuration

```
public class SecurityConfiguration {
```

@Bean

```
public SecurityFilterChain securityFilterChain(HttpSecurity http)
    throws Exception {
    return http.authorizeHttpRequests()
        .antMatchers(...antPatterns: "/login").permitAll()
        .antMatchers(...antPatterns: "/hebele").hasAuthority("READ")
        .antMatchers(...antPatterns: "/student/**").hasRole("USER")
        .antMatchers(...antPatterns: "/verySecretApi").denyAll()
        .anyRequest().authenticated()
        .and()
        .formLogin()
        .and()
        .build();
}
```

```
}
```




PRACTiCE

Uygulama

- ▶ /user ve /admin şeklinde iki rest endpoint oluşturun
- ▶ Admin ve user şeklinde iki user oluşturun, ve birine ADMIN yetkisini, diğerine USER yetkisini verin
- ▶ /user'a ADMIN veya USER yetkileri olan kişiler, /admin'e ise sadece ADMIN yetkisi olan kişiler erişebilir
- ▶ HttpSecurity'de uygun URL'lere uygun yetkilere sahip kişilerin erişebilmesini sağlayın
- ▶ Farklı kullanıcılar ile giriş yaparak uygun rest endpoint'lere uygun kullanıcı ile giriş yapabildiğinizi görün

@PreAuthorize

- ▶ Tüm URL'leri tek bir yerden kontrol etmek çok zor
- ▶ Doğrudan rest controller methodlarından access control sağlayabiliriz
- ▶ Configuration class'ımıza
- ▶ `@EnableGlobalMethodSecurity(prePostEnabled = true)` eklememiz gerekiyor

@Configuration

@EnableMethodSecurity

```
public class SecurityConfiguration {
```

@Bean

```
public SecurityFilterChain securityFilterChain(HttpSecurity http)
    throws Exception {
    return http.authorizeHttpRequests()
        .antMatchers(...antPatterns: "/login").permitAll()
        .antMatchers(...antPatterns: "/hebele").hasAuthority("READ")
        .antMatchers(...antPatterns: "/student/**").hasRole("USER")
        .antMatchers(...antPatterns: "/verySecretApi").denyAll()
        .anyRequest().authenticated()
        .and()
        .formLogin()
        .and()
        .build();
}
```

@RestController

```
public class TestController {
```

@GetMapping("/test")

@PreAuthorize("hasAuthority('READ')")

```
public String test() {
```

```
    return "Hello World!";
```

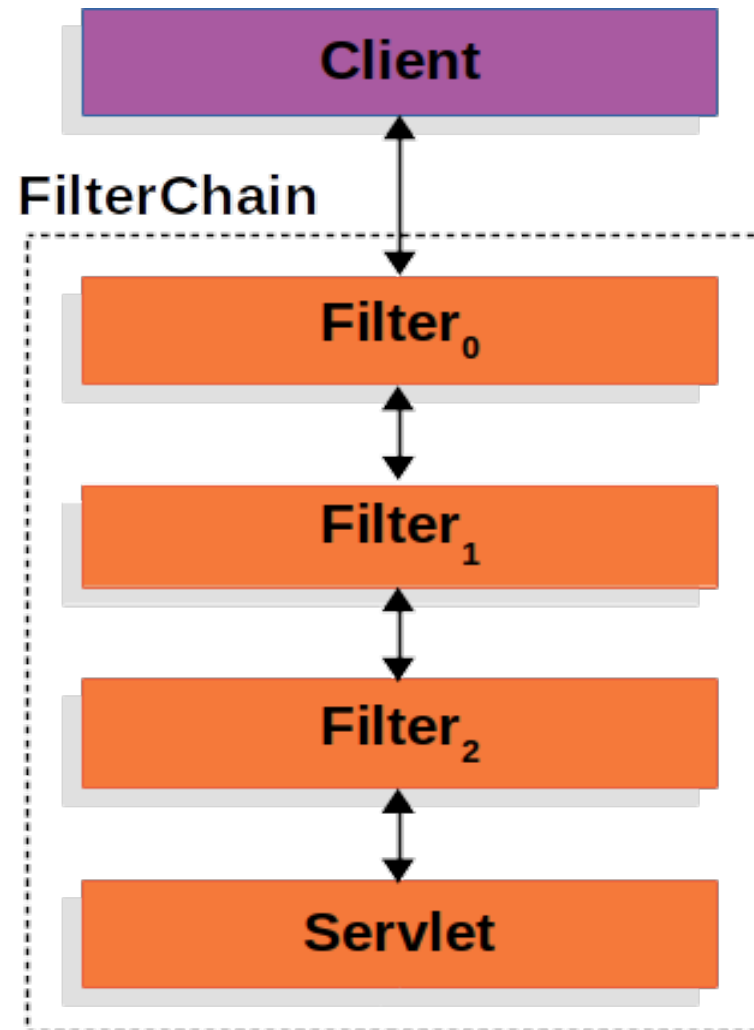
```
}
```

```
}
```

- ▶ Configuration, birden fazla URL için aynı şeyi tanımlamak için ideal
- ▶ @PreAuthorize, method seviyesinde ayrı yetkiler kullanmak için ideal

Authentication Mantığı ve Customization

Filter



```
public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain) {  
    // do something before the rest of the application  
    chain.doFilter(request, response); // invoke the rest of the application  
    // do something after the rest of the application  
}
```


Spring security filters

- ▶ Spring security'nin de kendine özgü filtreleri var
- ▶ Uygulama başlangıcında, bazı default filtreleri register ederek authentication ve authorization sağlar.
- ▶ Filter'lar, request'te bulunan bilgilere göre gerekli authentication mekanizmalarını çağırır

UserDetails ve UserDetailsService

- ▶ UserDetails, spring'in kullanıcıyı temsil etme yöntemi
- ▶ UserDetailsService, kullanıcıyı bir kaynaktan çekmek için kullanılan ana yöntem(LDAP, OpenID, veri tabanı vs.)
- ▶ Arkada tarafta spring UserDetailsService kullanarak kullanıcı bilgilerini çeker

```
@Service
public class CustomUserDetailsService implements UserDetailsService {

    @Override
    public UserDetails loadUserByUsername(final String username) {
        //Bir şekilde user'ı çekip, AuthenticationProvider'a döner
        return null;
    }
}
```

```
@AllArgsConstructor
public class CustomUser implements UserDetails {

    private String username;
    private String password;
    private List<SimpleGrantedAuthority> authorities;

    @Override
    public String getUsername() {
        return username;
    }

    @Override
    public String getPassword() {
        return password;
    }

    @Override
    public Collection<? extends GrantedAuthority> getAuthorities() {
        return authorities;
    }

    @Override
    public boolean isAccountNonExpired() {
        return true;
    }

    @Override
    public boolean isAccountNonLocked() {
        return true;
    }

    @Override
    public boolean isCredentialsNonExpired() {
        return true;
    }

    @Override
    public boolean isEnabled() {
        return true;
    }
}
```

```
@Configuration
@EnableMethodSecurity
public class SecurityConfiguration {

    @Autowired
    public SecurityConfiguration(AuthenticationManagerBuilder authenticationManagerBuilder,
                                CustomUserDetailsService customUserDetailsService)
        throws Exception {
        authenticationManagerBuilder
            .userDetailsService(customUserDetailsService)
            .passwordEncoder(NoOpPasswordEncoder.getInstance());
    }
}
```



PRACTiCE

Uygulama

- ▶ CustomUser ve CustomUserDetailsService şeklinde UserDetails ve UserDetailsService'i implement eden class yazın.
- ▶ CustomUserDetailsService, çağırıldığında kullanıcı adı ve şifresi aynı olan ve granted authority'leri boş olan bir CustomUser dönsün (içine koyduğu değerler, loadUserByUsername'den paslanan değerler olacak).
- ▶ Login ekranında kullanıcı adı ve şifreye aynı değeri yazarak giriş yapmayı deneyin.

AuthProvider

- ▶ Authentication işleminin asıl olduğu yer
- ▶ Bir security filter tarafında çağrılarak authentication yapılır(tam değil ama şimdilik böyle kalsın)

AuthProvider

javabrain.io

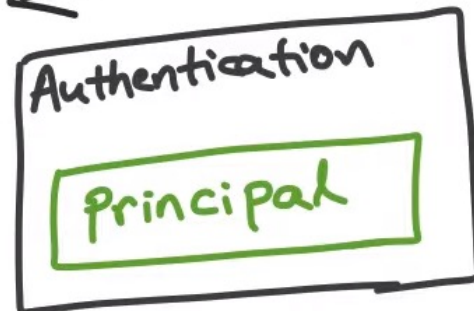


input



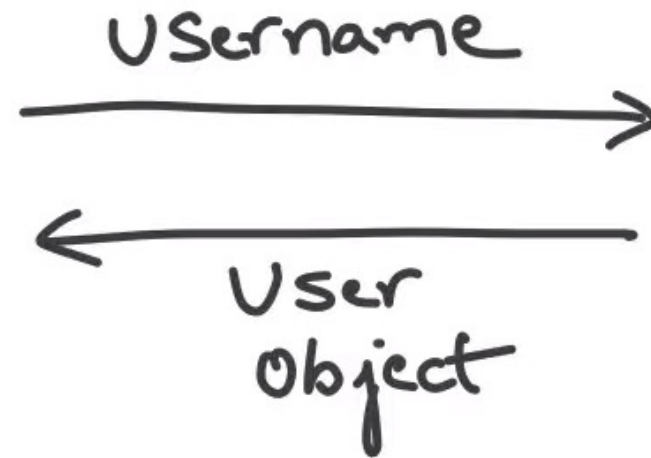
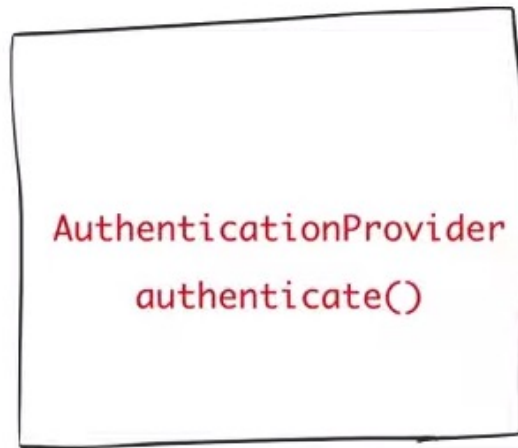
AuthProvider
authenticate()

output



Authentication

- ▶ Authentication işleminin yapılabilmesi için gerekli temel interface
- ▶ İçinde kullanıcı adı, şifre, principal, authorities gibi bilgileri tutar
- ▶ Default implementation olarak genelde UsernamePasswordAuthenticationToken class'ı kullanılır.



Spring Security app

I do
pwd auth

AuthenticationProvider
authenticate()

I do
OAuth

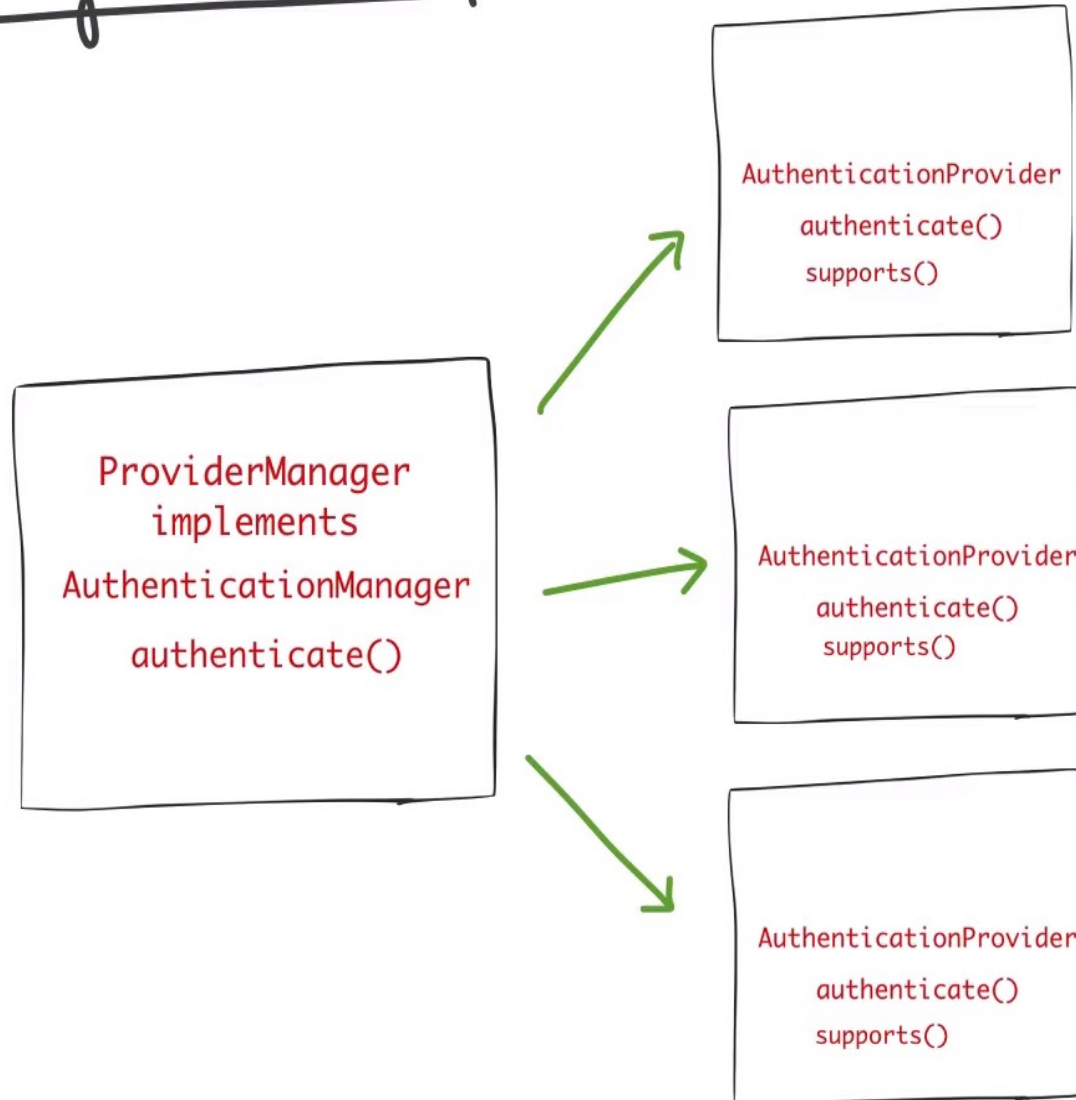
AuthenticationProvider
authenticate()

I do
LDAP auth

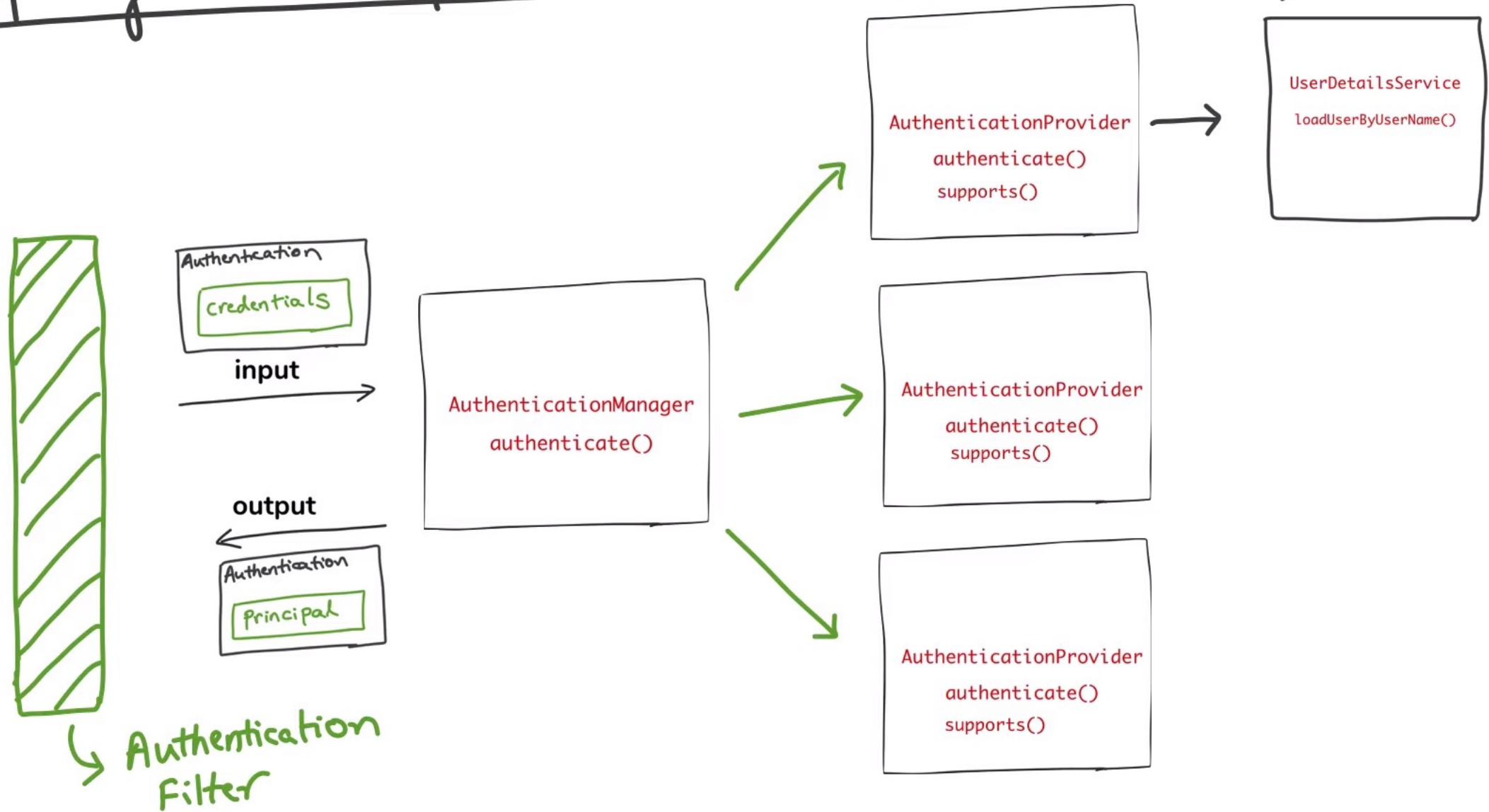
AuthenticationProvider
authenticate()

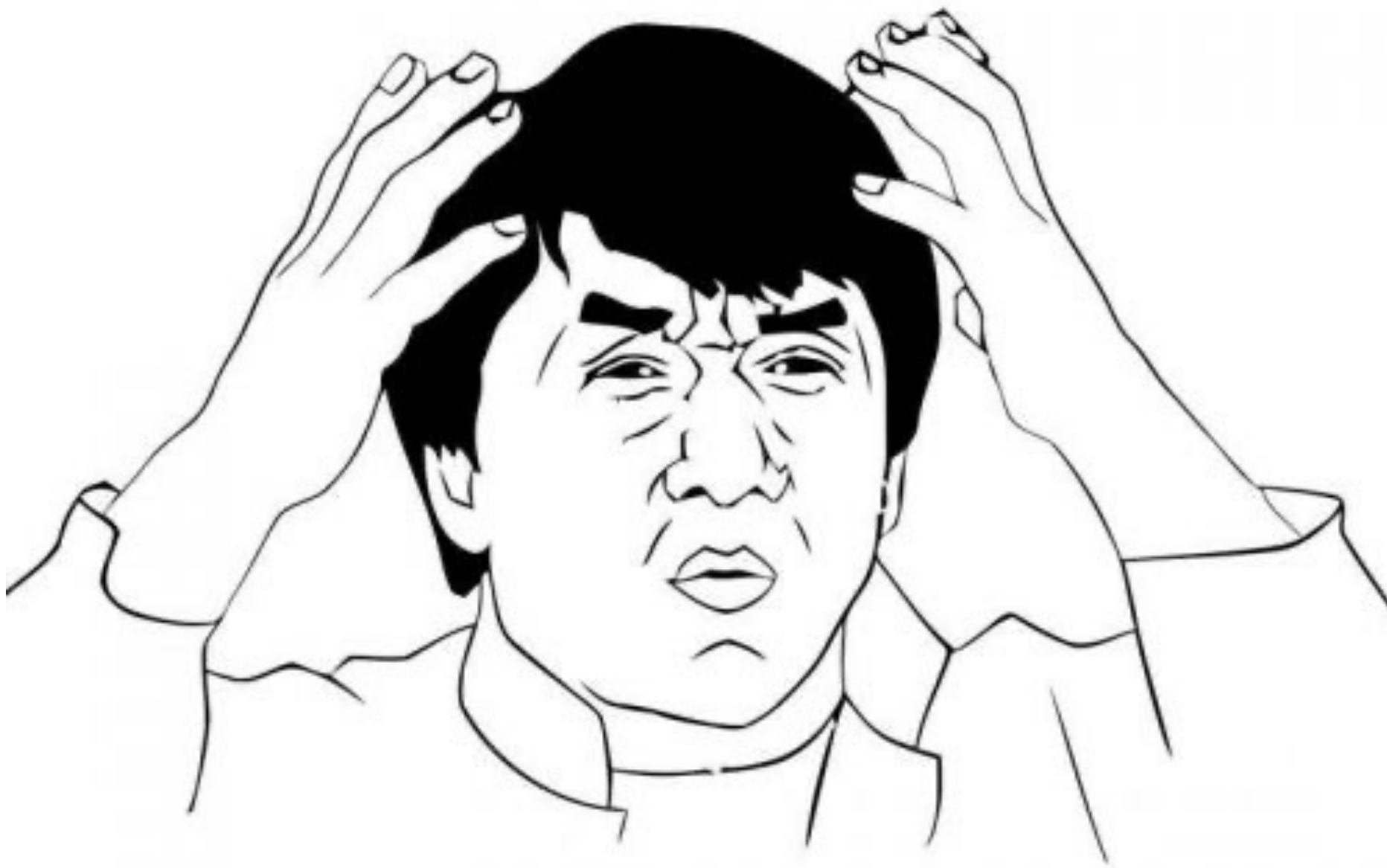
Spring Security app

javabrain.io



Spring Security app





Kullanıcı bilgilerini veri tabanına kaydetme

- ▶ Yazdığımız CustomUserDetailsService çok saçma bir iş yapıyor.
- ▶ UserDetailsService, kullanıcı bilgilerini veri tabanından almalı.
- ▶ Bunun için spring data repository'lerini ve entity'leri rahatlıkla kullanabiliriz.

```

@Entity
@Getter
@Setter
@NoArgsConstructor
@AllArgsConstructor
public class Users implements UserDetails {

    @Id
    @GeneratedValue
    private Long id;

    private String username;
    private String password;
    private boolean isAccountNonExpired;
    private boolean isAccountNonLocked;
    private boolean isCredentialsNonExpired;
    private boolean isEnabled;

    @ManyToMany(fetch = FetchType.EAGER, cascade = CascadeType.ALL)
    @JoinTable(name = "USER_AUTHORITIES",
        joinColumns = @JoinColumn(name = "USER_ID"),
        inverseJoinColumns = @JoinColumn(name = "AUTHORITY_ID"))
    private Set<Authority> authorities;

    @Override
    public boolean isAccountNonExpired() {
        return isAccountNonExpired;
    }

    @Override
    public boolean isAccountNonLocked() {
        return isAccountNonLocked;
    }

    @Override
    public boolean isCredentialsNonExpired() {
        return isCredentialsNonExpired;
    }

    @Override
    public boolean isEnabled() {
        return isEnabled;
    }
}

```

```

@Entity
@Data
public class Authority implements GrantedAuthority {

    @Id
    @GeneratedValue
    private Long id;

    @ManyToMany(mappedBy = "authorities")
    private Set<Users> userEntities;

    private String authority;
}

```

```

@Service
@RequiredArgsConstructor
public class UserDetailsServiceImpl implements UserDetailsService {

    private final UserRepository userRepository;

    @Override
    public UserDetails loadUserByUsername(final String username) {
        return userRepository.findByUsername(username);
    }
}

```

```

public interface UserRepository extends JpaRepository<Users, Long> {
    Users findByUsername(String username);
}

```




PRACTiCE

Uygulama

- ▶ /user ve /admin şeklinde iki rest endpoint oluşturun
- ▶ Admin ve user şeklinde iki user oluşturun, ve birine ADMIN yetkisini, diğerine USER yetkisini verin
- ▶ /user'a ADMIN veya USER yetkileri olan kişiler, /admin'e ise sadece ADMIN yetkisi olan kişiler erişebilir
- ▶ HttpSecurity'de uygun URL'lere uygun yetkilere sahip kişilerin erişebilmesini sağlayın
- ▶ Farklı kullanıcılar ile giriş yaparak uygun rest endpoint'lere uygun kullanıcı ile giriş yapabildiğinizi görün

Custom AuthenticationProvider

- ▶ Eğer kendi authentication mantığımızı oluşturmak istiyorsak, AuthenticationProvider'ı implement etmemiz gerekiyor

SecurityContextHolder ve SecurityContext

- ▶ Authentication başarılı bir şekilde gerçekleştiğinde, authentication objesi, uygulama tarafından rahatça erişilebilmesi için SecurityContext objesine konur.

SecurityContextHolder

SecurityContext

Authentication

Principal

Credentials

Authorities

```
@RestController
public class TestController {

    @GetMapping("/test")
    public void test() {
        SecurityContext context = SecurityContextHolder.getContext();
        Authentication authentication = context.getAuthentication();
        UserDetails userDetails = (UserDetails) authentication.getPrincipal();
        System.out.println(userDetails.getUsername());
        System.out.println(userDetails.getPassword());
    }
}
```

