برنامه‌سازی وب (۴۰۴۱۹)

مقدمه‌ای بر پیاده‌سازی مکانیزم‌های امنیتی

دانشکده‌ی مهندسی کامپیوتر دانشگاه صنعتی شریف

مدرس : یحیی پورسلطانی (دانشجوی دکترای تخصصی مهندسی کامپیوتر)

بهار ۱۴۰۴

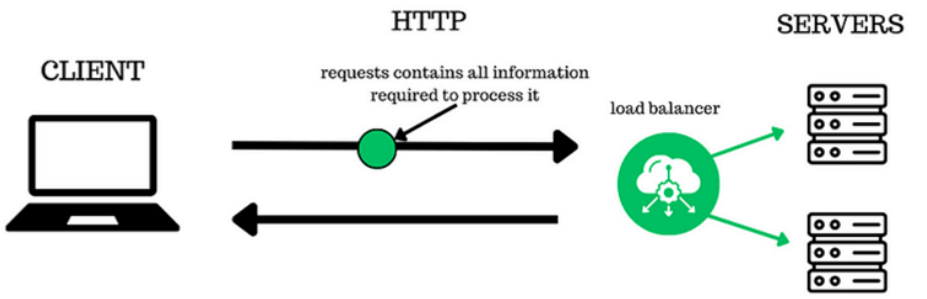
2

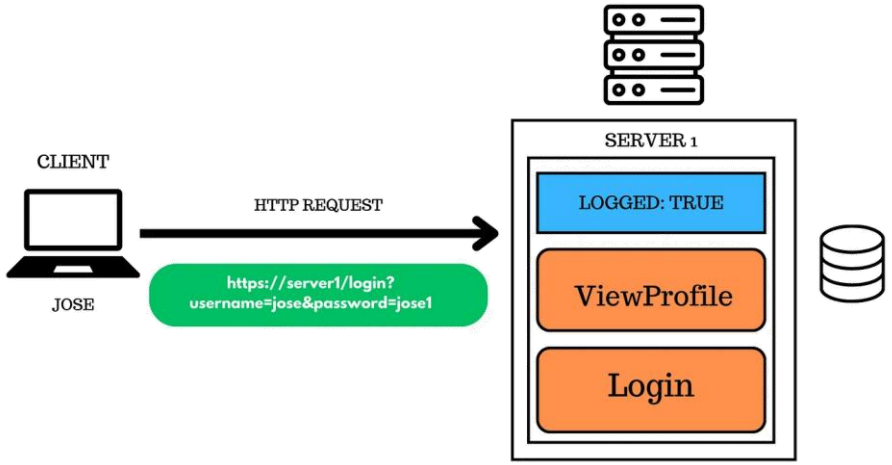
از امنیت بگو!

امنیت

- درز اطلاعات از زمان‌های خیلی قدیمی (حتی جنگ جهانی) موضوع مهمی بوده.
- اطلاعات بر بستر شبکه جابجا می‌شوند. بنابراین باید ملاحظاتی را در خصوص روش ارسال و دریافت اطلاعات در نظر گرفت.
  - اطلاعات بایستی به طریق درستی در مبداء رمزگذاری شده و در مقصد رمزگشایی شود.
  - باید از روش مناسب برای ارسال و دریافت داده‌ها استفاده نمود.
    - استفاده از Session
    - استفاده از کوکی

Statelessness!





- یکی از ویژگی‌های پروتکل Http : Stateless
  - از اطلاعات درخواست‌های قبلی بی اطلاع است!
  - چالش جدی در نگهداشت اطلاعات کاربران در گذر زمان
    - کدام یک از کاربران درحال حاضر احراز هویت شده اند؟
    - آیا درخواست واصله، از جانب یک کاربر مجاز ارسال شده؟
- راهکار : افزودن اطلاعات جانبی به درخواست HTTP
- این موضوع چه مشکلاتی را می‌تواند به همراه داشته باشد؟

چگونگی نگهداشت اطلاعات درخواست‌های قبلی

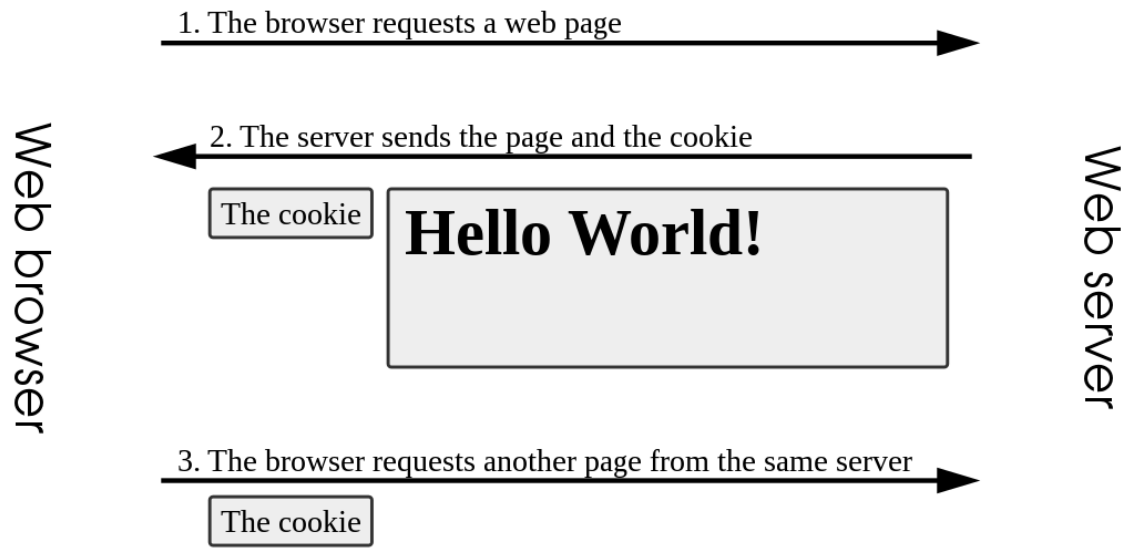
- یک مثال واقعی : فرض کنید که در یک مطب پزشک، اطلاعات مراجعات قبلی بیماران در قالب پرونده ایشان ذخیره می‌شود و هر پرونده، شماره دارد. سه روش برای ذخیره‌سازی چیشنهاد می‌شود :
  - اول – پرونده در دست بیمار باشد و هر مرتبه و در هر مراجعه باید آن را به همراه بیاورد.
    - مشابه با ارسال و دریافت در کوکی.
  - دوم – پرونده در مطب باشد و بیمار با هر بار مراجعه، باید شماره پرونده خود را اطلاع دهد تا بازیابی شود!
    - مشابه با نگهداری یک شناسه به نشست (سوابق قبلی) و دریافت آن در کوکی

نشست (Session)

- یک نشست شامل تمامی اطلاعاتی است که کاربر در جریان تعامل با سرور آن را نگه می‌دارد.
  - برای برقراری پیوستگی در جریان تعامل با سرور.
- تا زمانی که نشست معتبر باشد، کاربر نیز امکان تعامل با سیستم را دارد.
- اما چگونه آن را می‌توان نگه داشت؟
  - راهکار خیلی ساده – آن را به سمت کاربر ارسال کرد و سپس دریافت نمود!

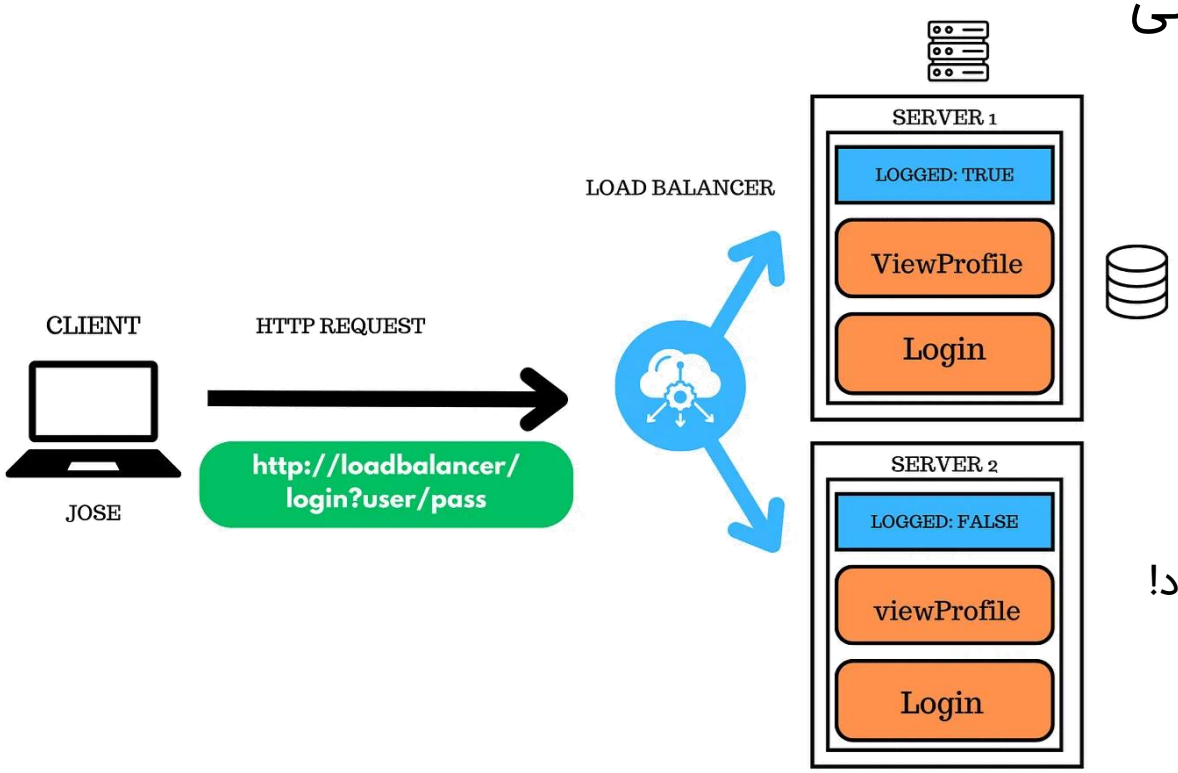
ارسال و دریافت اطلاعات با Cookie

- ساده‌ترین روش ممکن برای ارسال و دریافت اطلاعات، قرار دادن آن در بخشی از پروتکل **Http** است که آن را **Cookie** می‌نامیم (مبتنی بر **Key-Value**) است.
- اما آیا قرار دادن تمامی اطلاعات (اطلاعات نشست) در کوکی کار درستی است؟
  - خیر – به طور کامل قابلیت شنود دارد.



چالش ذخیره و بازیابی اطلاعات نشست‌ها

- ذخیره‌ی اطلاعات نشست‌ها علاوه بر چالش‌های امنیتی ، مستلزم مصرف منابع بسیاری است!
  - نشست‌های منقضی باید از حافظه‌ی سمت سرور پاک شوند.
  - محل ذخیره‌سازی نشست :
    - راهکار اول – کل آن در کوکی ذخیره شود!
    - راهکار دوم – در سمت سرور و در حافظه‌ی اصلی ذخیره شود!
    - بازهم چالش جابجایی رخ می‌دهد!

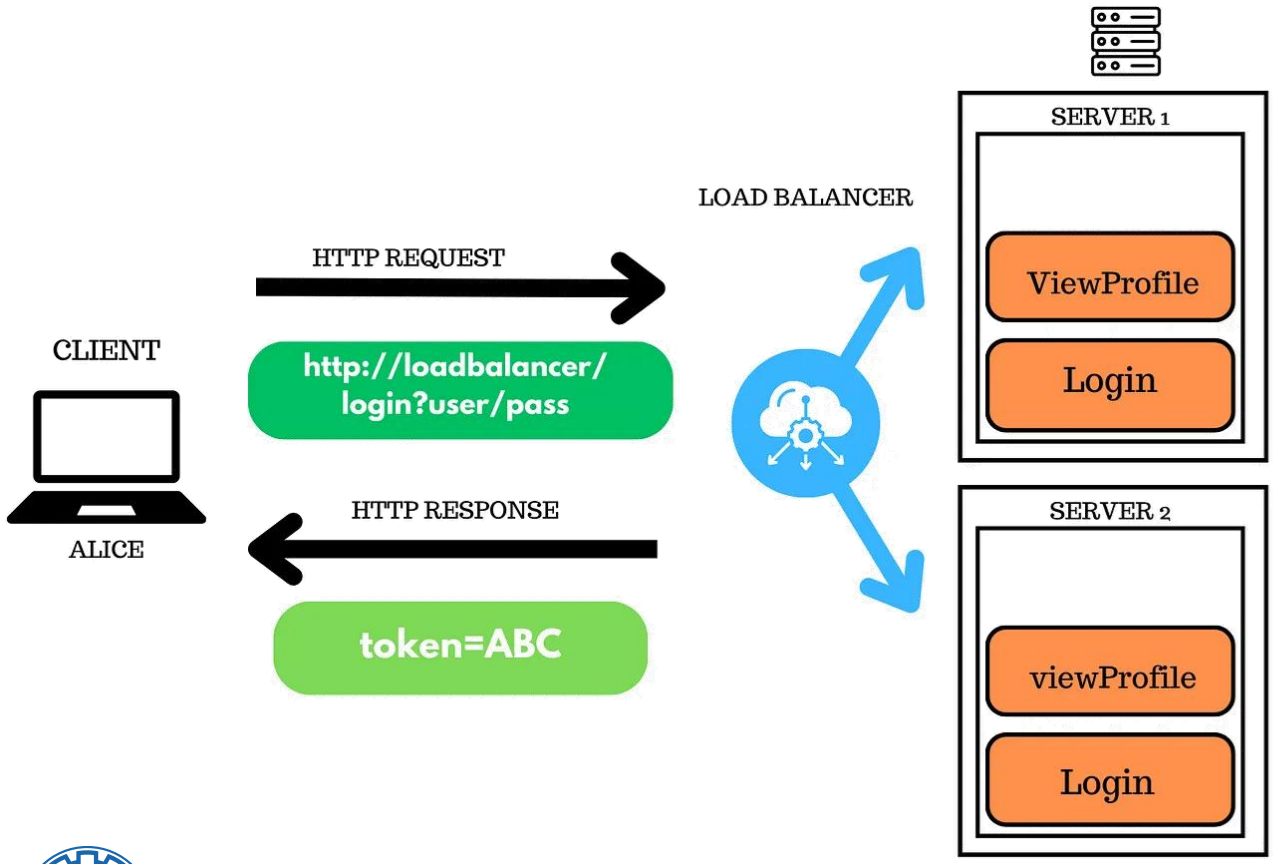


https://medium.com/@martinezmendietagerman/scaling-web-apps-stateful-vs-stateless-87ce157b8dcf

https://medium.com/@martinezmendietagerman/scaling-web-apps-stateful-vs-stateless-87ce157b8dcf

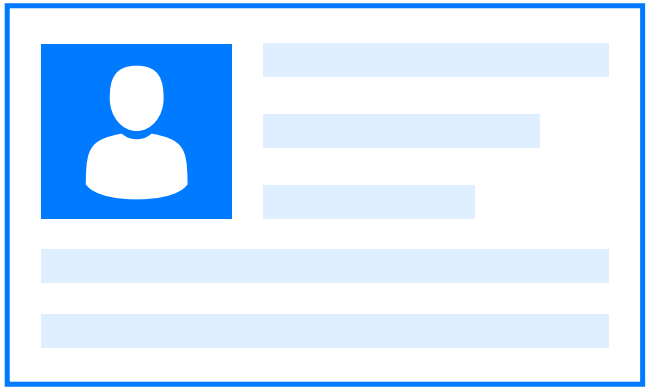
راهکار پذیرفته شده

- ذخیره‌ی نشست‌ها در سمت سرور و در یک پایگاه داده‌ی سریع (Redis, Hazlecast و...)
- ارسال و دریافت اطلاعات در قالب رد و بدل کردن شناسه‌ی نشست.
- حذف نشست‌های منقضی شده از سمت سرور.



Authentication vs Authorization

- مفهوم Authentication : چه کسی می‌تواند از سیستم استفاده کند؟
- مفهوم Authorization : هر یک از کاربران سیستم، از چه بخش‌هایی از آن می‌توانند استفاده کنند؟



Authentication

Who you are

Authorization

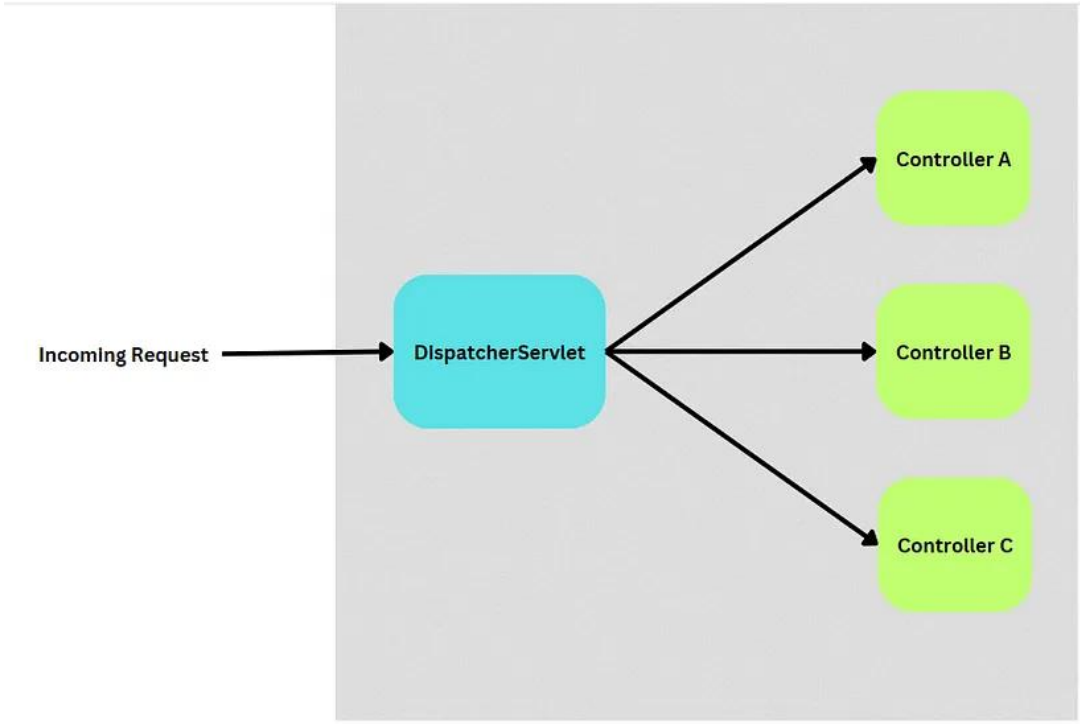
What you can do

11

برقراری امنیت در برنامه

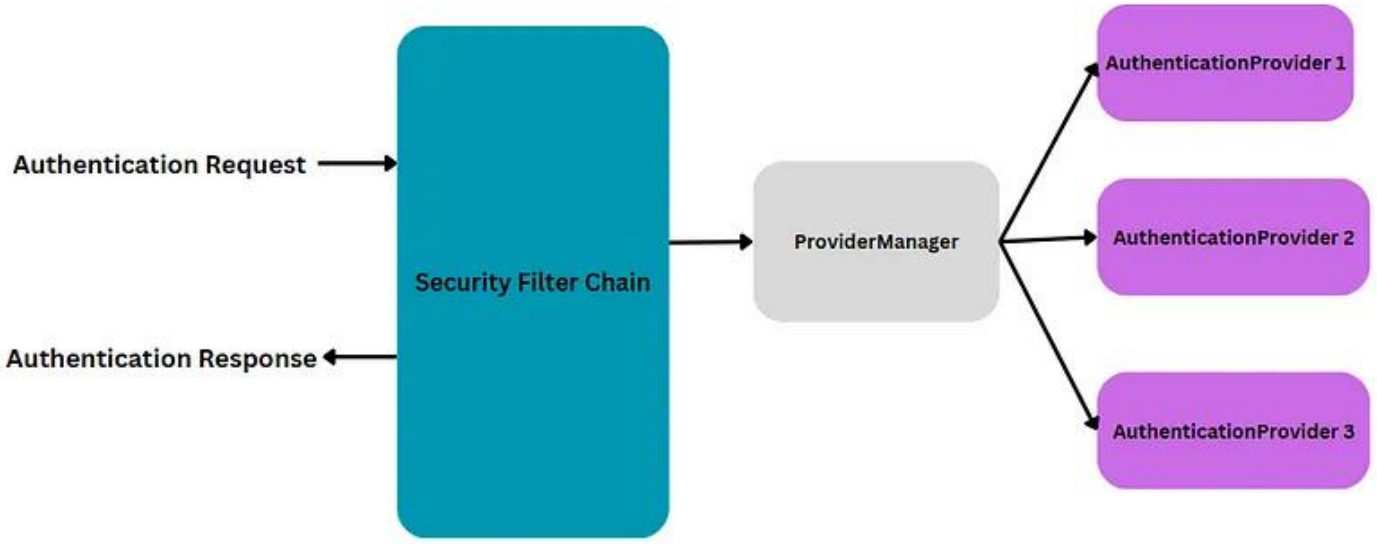
مکانیزم بدون امنیت!

- در حالتی که هیچ مکانیزم امنیتی بر روی برنامه برقرار نباشد، تمامی آدرس‌ها در دسترس بوده و برای همه‌ی کاربران قابل استفاده هستند.
  - تمامی کنترلرها در دسترس هستند!



چگونگی برقراری امنیت در Spring Security

- برای احراز هویت یک کاربر در یک برنامه‌ی Spring Boot ، مکانیزم‌های مختلفی وجود دارد که یکی از بدیهی ترین آن‌ها، دریافت نام کاربری و رمز عبور و تخصیص یک توکن به کاربر است.

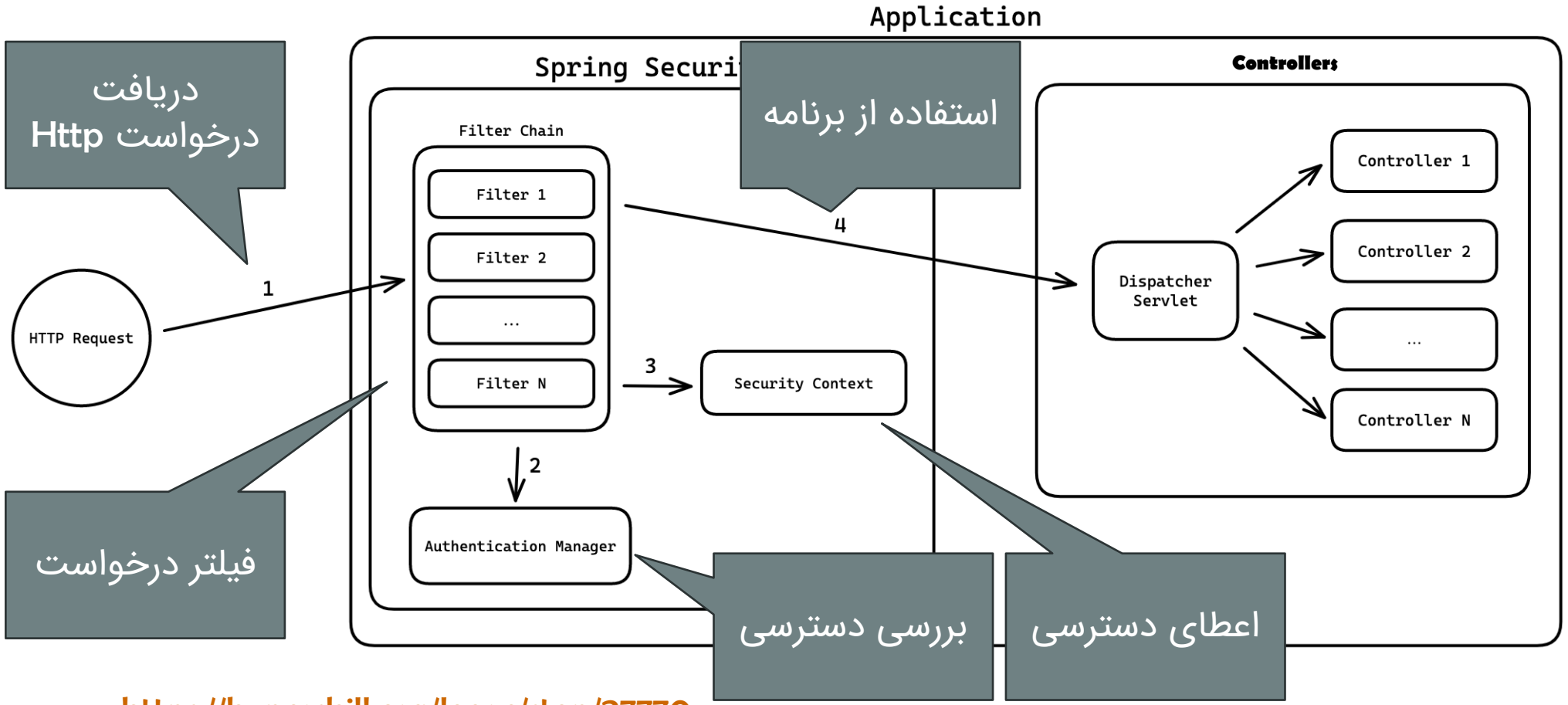


1. دریافت درخواست
2. فیلتر درخواست
3. احراز هویت کاربر با یکی از روش‌های ممکن.
4. قرار دادن اطلاعات کاربر احراز شده در Security Context .

https://medium.com/@aprayush20/understanding-spring-security-authentication-flow-f9bb545bd77

14 مکانیزم و معماری Spring Security

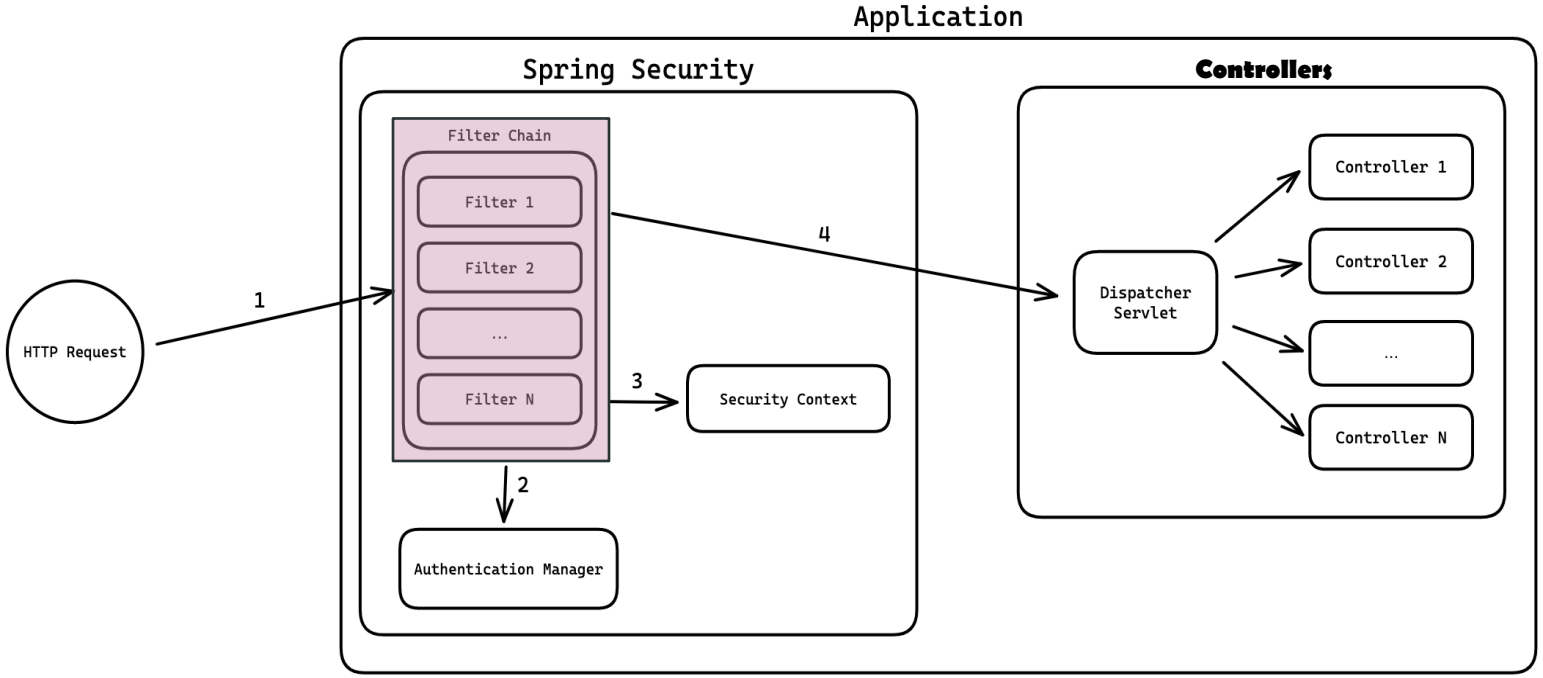
روند (کلی) احراز هویت یک درخواست



https://hyperskill.org/learn/step/27770

فیلتر درخواست‌های ورودی

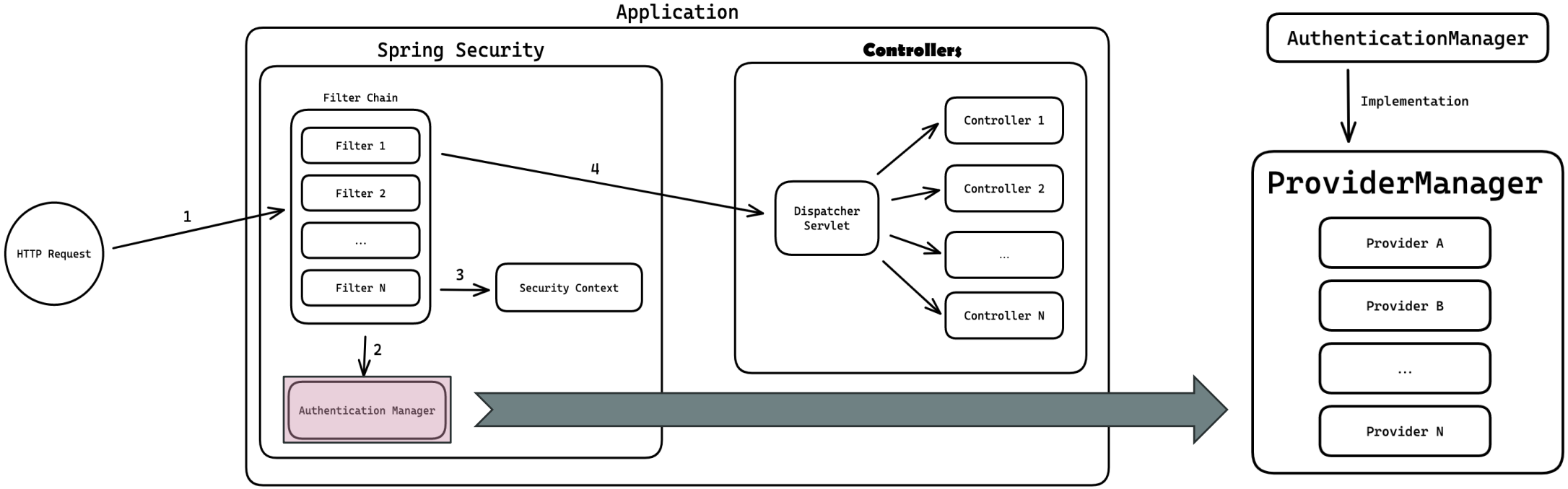
- درخواست ورودی پس با گذشت از فیلترهای امنیتی، از نظر مجوزهای دسترسی بررسی می‌شود.
- فیلترها دارای ترتیب هستند.
- پس از بررسی فیلترها، درخواست به Authentication Manager منتقل می‌شود.



https://hyperskill.org/learn/step/27770

روش‌های احراز هویت با Authentication Manager

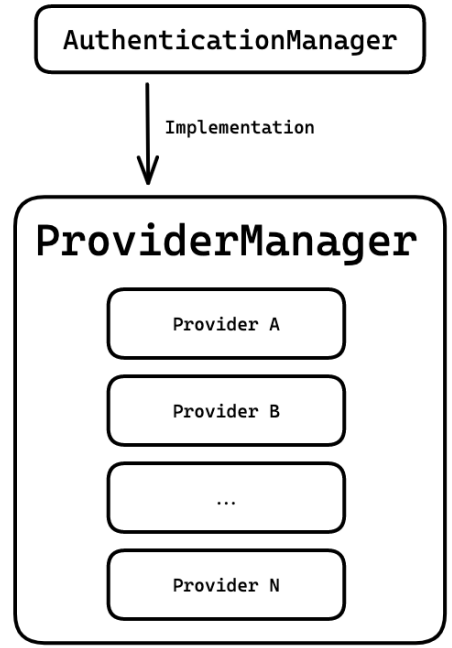
- روش‌های مختلفی با استفاده از واسط Authentication Manager پیاده‌سازی شده‌است.
- پیاده‌سازی پیش فرض : Provider Manager شامل روش‌های ممکن برای احراز هویت است.



https://hyperskill.org/learn/step/27770

روش‌های احراز هویت با Authentication Manager

- روش‌های مختلفی با استفاده از واسط Authentication Manager پیاده‌سازی شده‌است.
- پیاده‌سازی پیش فرض : Provider Manager شامل روش‌های ممکن برای احراز هویت است.



روش‌های مختلف احراز هویت را پیاده‌سازی کرده‌است:

1. با استفاده از Dao Authentication Provider
2. با استفاده از Jaas Authentication Provider
3. با استفاده از Open Id
4. با استفاده از OAuth2

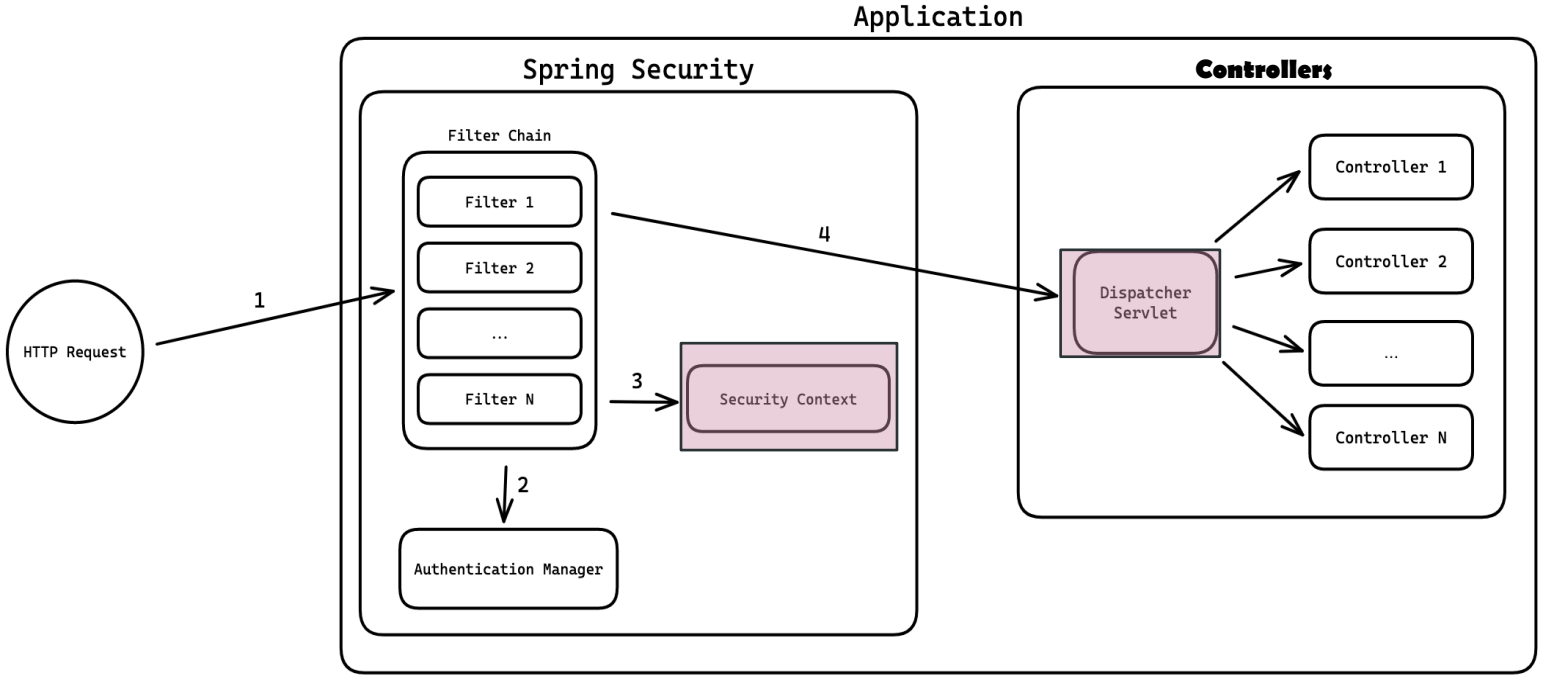
استخراج اطلاعات کاربری از روی حافظه

User Details Service

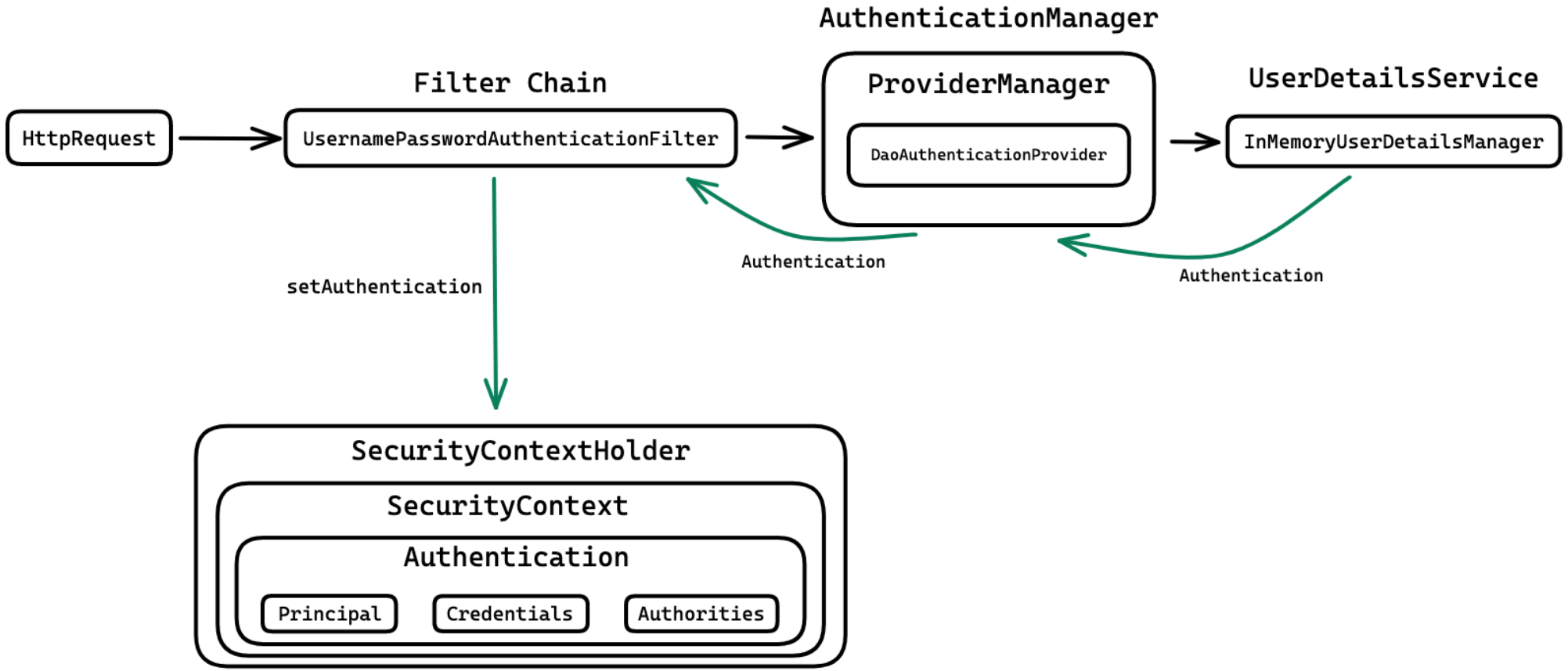
https://hyperskill.org/learn/step/27770

احراز هویت و قرارگیری اطلاعات کاربر در Security Context

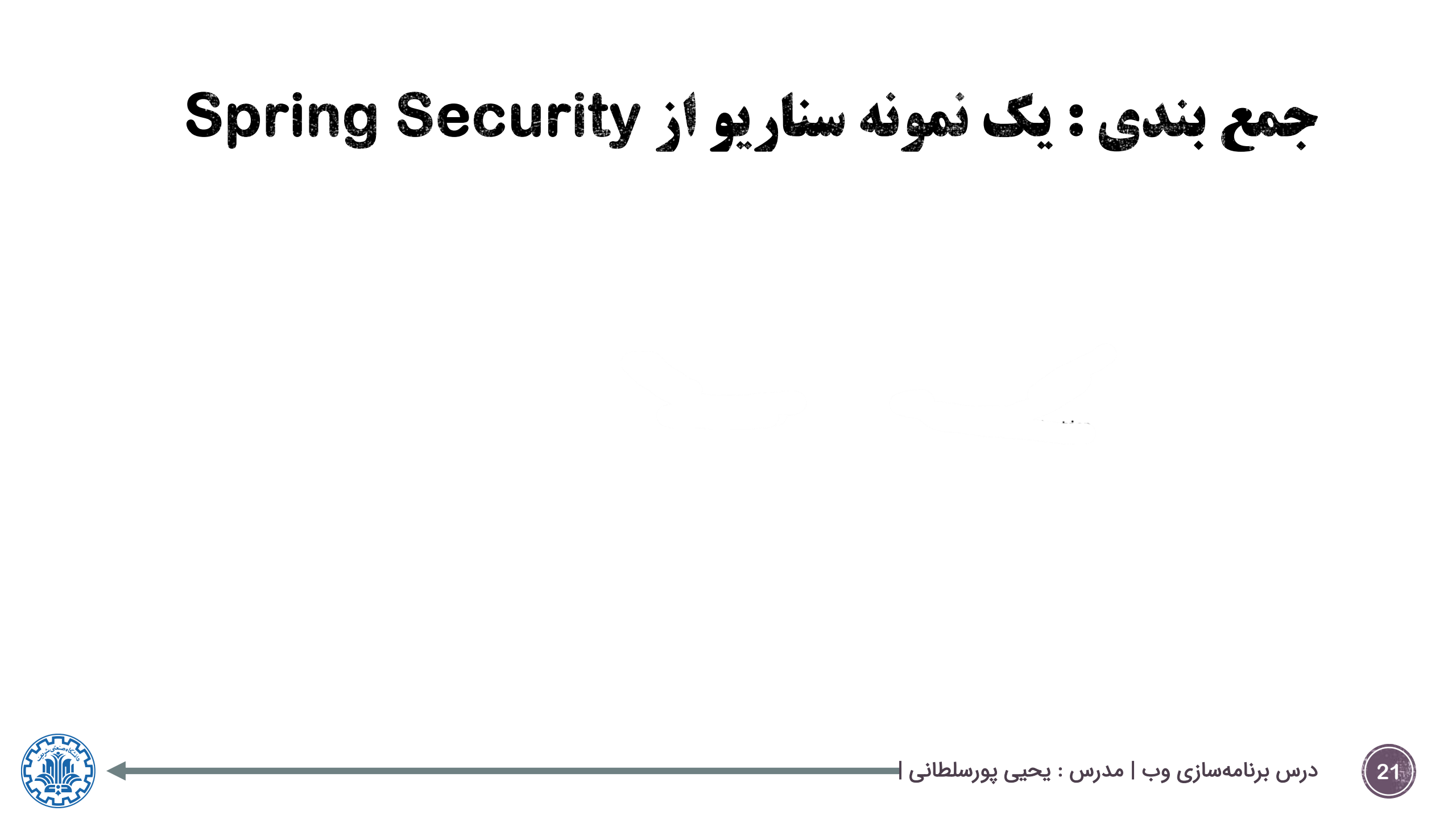
- محل قرارگیری اطلاعات کاربران احراز هویت شده در Spring Security Context است.
- پس از احراز هویت کاربران، اطلاعات ایشان در قالب یک شی از نوع User Details در Spring Security Context قرار می‌گیرد.
- در نهایت کاربر احراز هویت شده می‌تواند بر اساس مجوزها از برنامه استفاده کند.



جمع بندی : یک نمونه سناریو از Spring Security



جمع بندی : یک نمونه سناریو از Spring Security



22 نصب و به کارگیری Spring Security

آماده‌سازی Spring Security

- برای آن که بتوان از **Spring Security** استفاده نمود، باید **maven Dependency** آن را اضافه کرد. در این صورت یک کاربر با نام کاربری پیش فرض **user** و رمز عبور پیش فرض ساخته میشه.
- در این صورت تمامی مسیرها بسته می‌شود!
  - چه باید کرد؟
    - تغییر فیلترها
    - تنظیم سرویس **User Details Service**

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

پیکره‌بندی اولیه

- برای آن که تنظیمات پیشفرض **Spring Security** غیر فعال شود، لازم است که یک کلاس پیکره بندی تعریف شود و ضمن تعریف **Bean**های مورد نیاز برای ساخت **User Details** ، یک **Bean** برای رمزنگاری نیز تعریف شود.

```
@Configuration
@EnableWebSecurity
public class SecurityConfig {


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

}
```

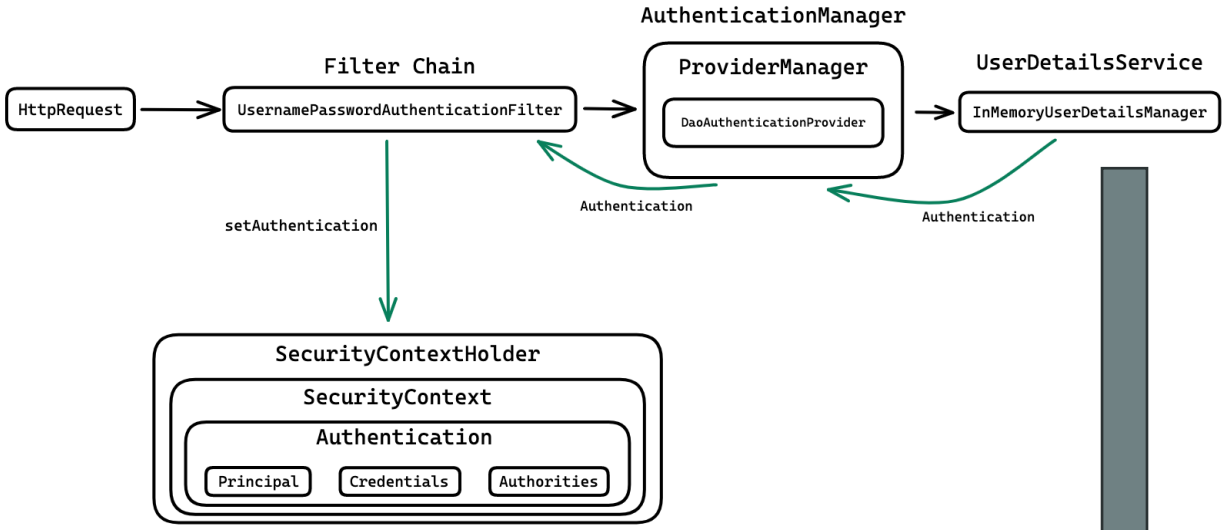
پیکره بندی User Details Service

```
@Configuration
@EnableWebSecurity
public class SecurityConfig{

   @Bean
   public UserDetailsService users() {
      UserDetails user = User.builder()
            .username("ali")
            .password(passwordEncoder().encode("123"))
            .roles("USER")
            .build();
      UserDetails admin = User.builder()
            .username("admin")
            .password(passwordEncoder().encode("123"))
            .roles("USER", "ADMIN")
            .build();
      return new InMemoryUserDetailsManager(user, admin);
   }

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

- پیکره‌بندی را باید تغییر داد!
- اولین راهکاری که به ذهن می‌رسد این هست که کاربرانی را اضافه کنیم تا بتوانیم از طریق ایشان نیز به سیستم وارد شویم!
- نیازمند پیکره‌بندی خاص است!
- روشی برای بارگزاری کاربران وجود داشته‌باشد.

ساخت User Details از روی اطلاعات پایگاه داده

- در روش قبل، اطلاعات کاربری به صورت ثابت و در حافظه‌ی اصلی ساخته می‌شد.
  - ترجیح بر این است که این اطلاعات بر اساس مشخصات کاربران سیستم (که بر روی پایگاه داده قرار گرفته‌است) تشکیل شود.
  - بنابراین لازم است که سرویس مخصوص User Details را به گونه‌ای بسازیم که امکان بارگزاری اطلاعات کاربران از پایگاه داده به Security Context مقدور باشد.



دریافت اطلاعات از پایگاه داده

```
@Service
public class SecurityService implements UserDetailsService {
    @Autowired
    CustomerRepository customerRepository;

    @Override
    public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {
        Optional<Customer> customerOptional = customerRepository.findByUsername(username);
        if(customerOptional.isPresent()){
            Customer customer = customerOptional.get();

            return User.withUsername(username)
                    .username(customer.getUsername())
                    .password(customer.getPassword())
                    .roles("USER").build();
        }else{
            throw new UsernameNotFoundException("Can not find username with name " + username);
        }
    }
}
```

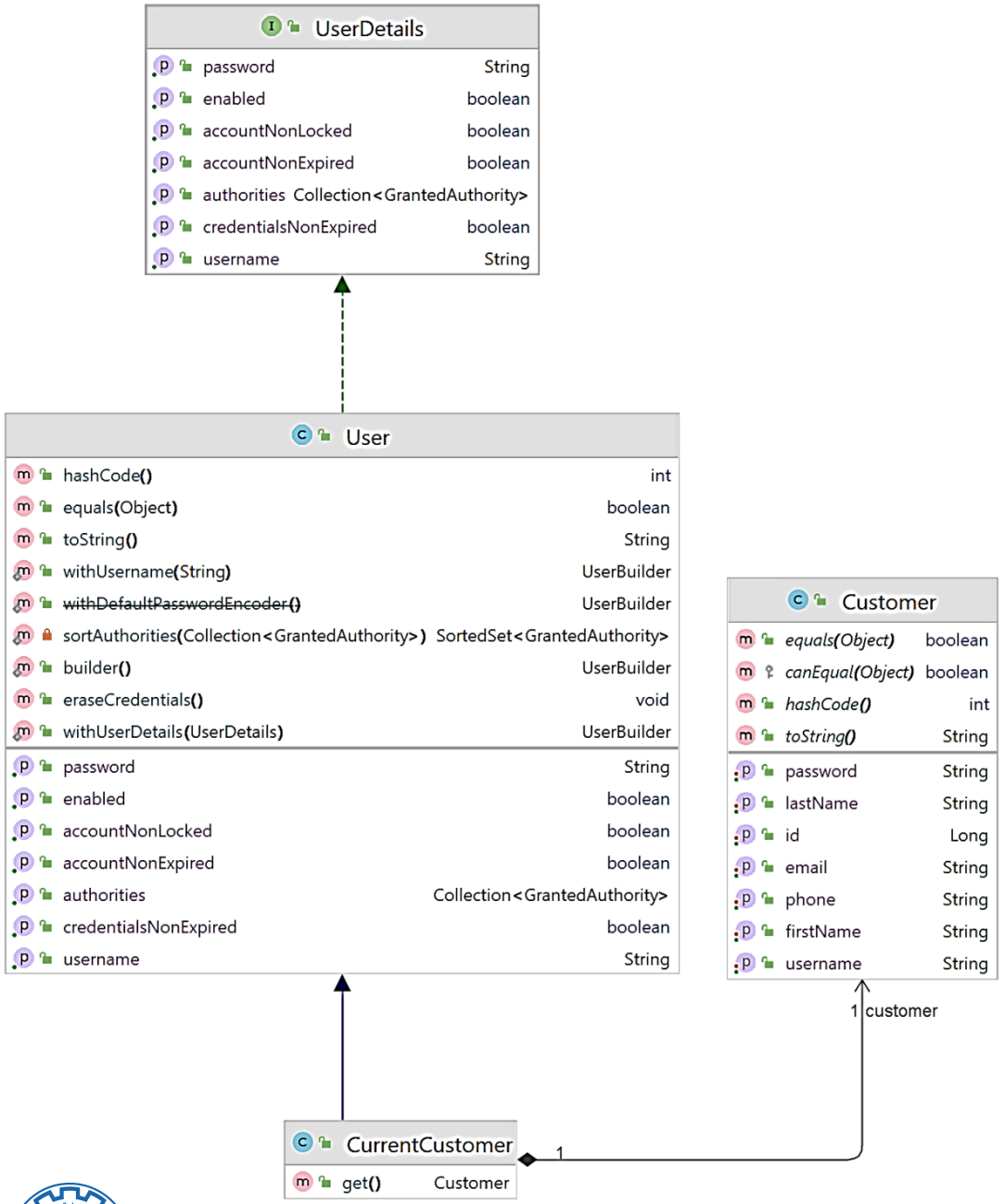
توسعه‌ی سرویس User Details Service و پیاده‌سازی متد load By Username

استخراج کاربر بر اساس username و ساخت User Details از روی آن

توسعه‌ی User Details

- اطلاعات کاربران احراز هویت شده در قالب اشیائی از نوع User Details توسط Spring Security Context فهمیده می‌شوند.
- اطلاعات محدودی دارد :
  - User name
  - Password
  - Role
- برای آن که سایر اطلاعات کاربری نیز به این Object اضافه شود، بایستی متناسب با نیازمندی کلاس User Details توسعه داده شود (extent) و توسط User Details Service به Security Context داده شود.

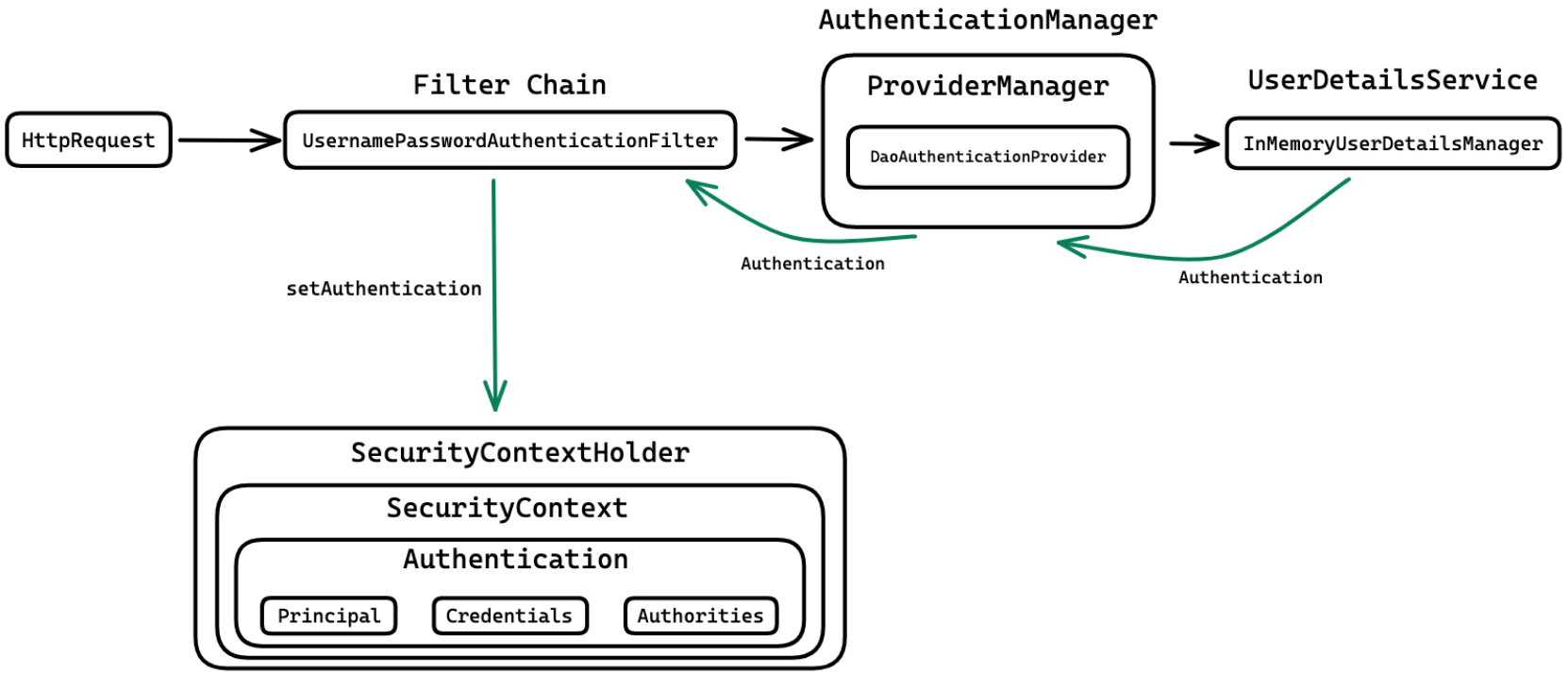
توسعه‌ی User Details




```
public UserDetails loadUserByUsername(String username) throws
UsernameNotFoundException {
      Optional<Customer> customerOptional =
customerRepository.findByUsername(username);
      if(customerOptional.isPresent()){
         Customer customer = customerOptional.get();

return new      CurrentCustomer(customer,customer.getUsername()
                        ,customer.getPassword(),
                        AuthorityUtils.createAuthorityList("ROLE_USER"));
      }else{
         throw new UsernameNotFoundException("...");
      }
   }
```

دسترسی به اطلاعات کاربر احراز شده

- پس از قرارگیری اطلاعات کاربر احراز هویت شده در قالب آبجکت Principal در دسترس است.



31 فیلتر درخواست‌ها

فیلتر درخواست‌ها

- هرگونه درخواست دریافت شده توسط برنامه، باید دنباله‌ای از فیلترها را بگذراند تا بتواند هم احراز شود و هم Authorize شود.
- مثال :
  - چه مسیرهایی باز است؟
  - چه مسیرهایی باید توسط یک نقش خاص مورد استفاده قرار گیرد؟
  - آیا امکان ورود از طریق فرم ورود مقدور است؟
  - در صورت ورود (یا خروج) موفقیت آمیز، چه اتفاق دیگری رخ دهد؟

مثالی از فیلتر درخواست‌ها

```
@Bean
public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {
    http
        .authorizeRequests()
            .requestMatchers("/api/auth/**").permitAll()
            .requestMatchers( "/users/**").authenticated()
            .anyRequest().authenticated()
        .and()
        .formLogin();

    return http.build();
}
```

ساخت یک Endpoint مقدماتی برای ورود و خروج

```
@PostMapping("/login")
public ResponseEntity<Object> login(@RequestBody LoginRequest loginRequest, HttpServletRequest request) {
   try {
      Authentication authentication = authenticationManager.authenticate(
            new UsernamePasswordAuthenticationToken(loginRequest.getUserName(), loginRequest.getPassword()));
      SecurityContextHolder.getContext().setAuthentication(authentication);
      return ResponseEntity.ok(request.getSession().getId());
   } catch (AuthenticationException e) {
      return ResponseEntity.status(HttpStatus.UNAUTHORIZED).body("Login failed: " + e.getMessage());
   }
}

@PostMapping("/logout")
public ResponseEntity<String> logout(HttpServletRequest request) {
   request.getSession().invalidate();
   return ResponseEntity.ok("Logout successful");
}
```

35

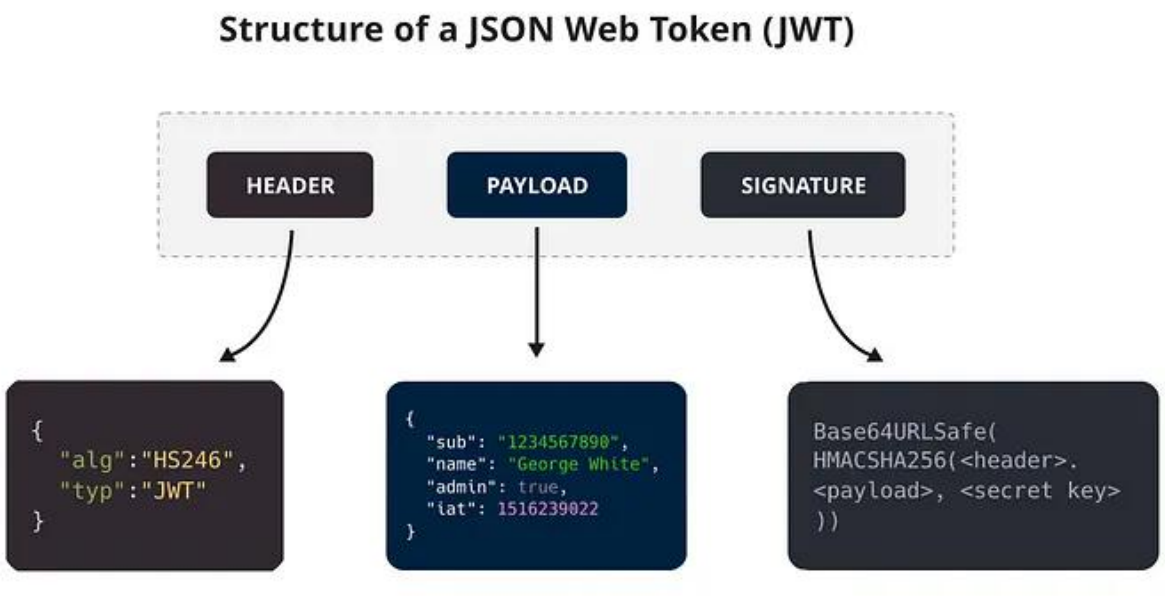
آشنایی با JWT

تا الان ...

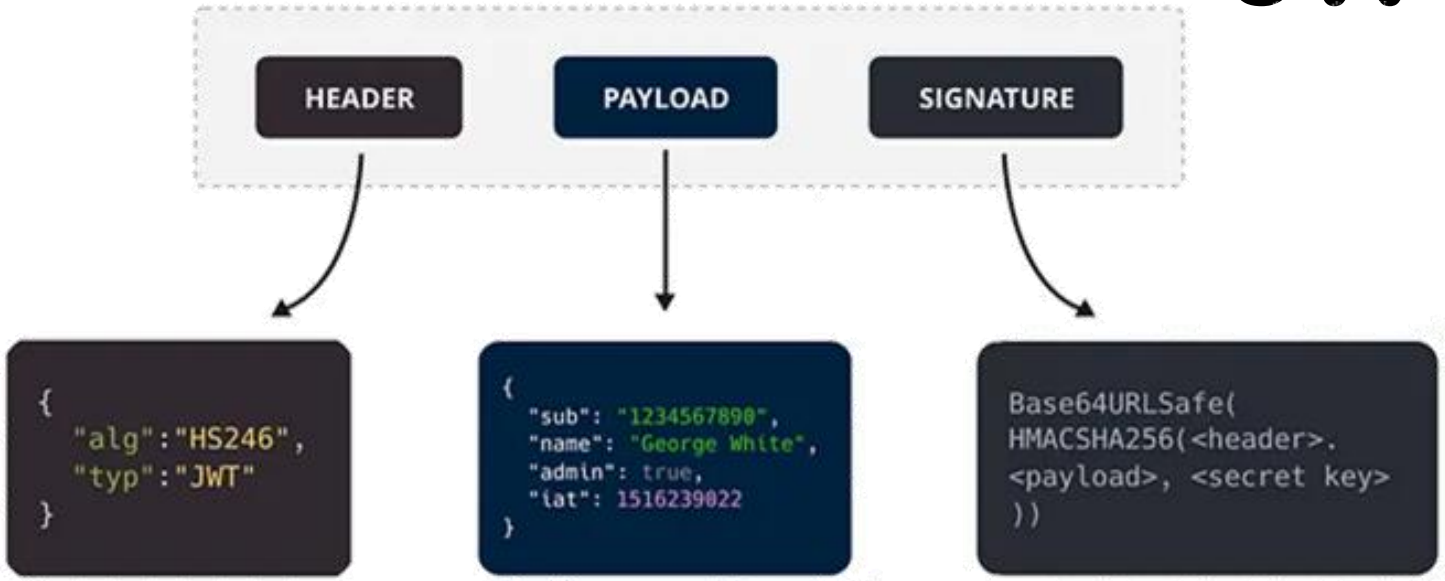
- تا اینجا با ساختن یک مکانیزم مقدماتی برای احراز هویت و کنترل دسترسی آشنا شدیم.
  - برای ورود و خروج، از صفحه‌ی لاگین پیش فرض استفاده کردیم.
- سوال : اگر بخواهیم با هر کلاینتی، نظیر وب، موبایل و.... بتوانیم از سیستم استفاده کنیم، این روش کار آمد است؟
  - در بسیاری از مواقع خیر! چون در خیلی از این ابزارها استفاده از **cookie** به راحتی ممکن است مقدور نباشد.
  - همچنین با جدا شدن کلاینت از سرور، ممکن است چالش‌های امنیتی جدید، نظیر مدیریت **CORS** برای ما دشوار باشد.

احراز هویت و کنترل دسترسی با JWT

- منظور از JWT ، خلاصه شده‌ی Java Web Token است.
- یک روش احراز هویت نسبتا امن تر و مستقل از نوع کلاینت است.
- روش تامین امنیت : اطلاعات درخواست در مبدا رمزگذاری شده و در مقصد رمزگشایی می‌شود.

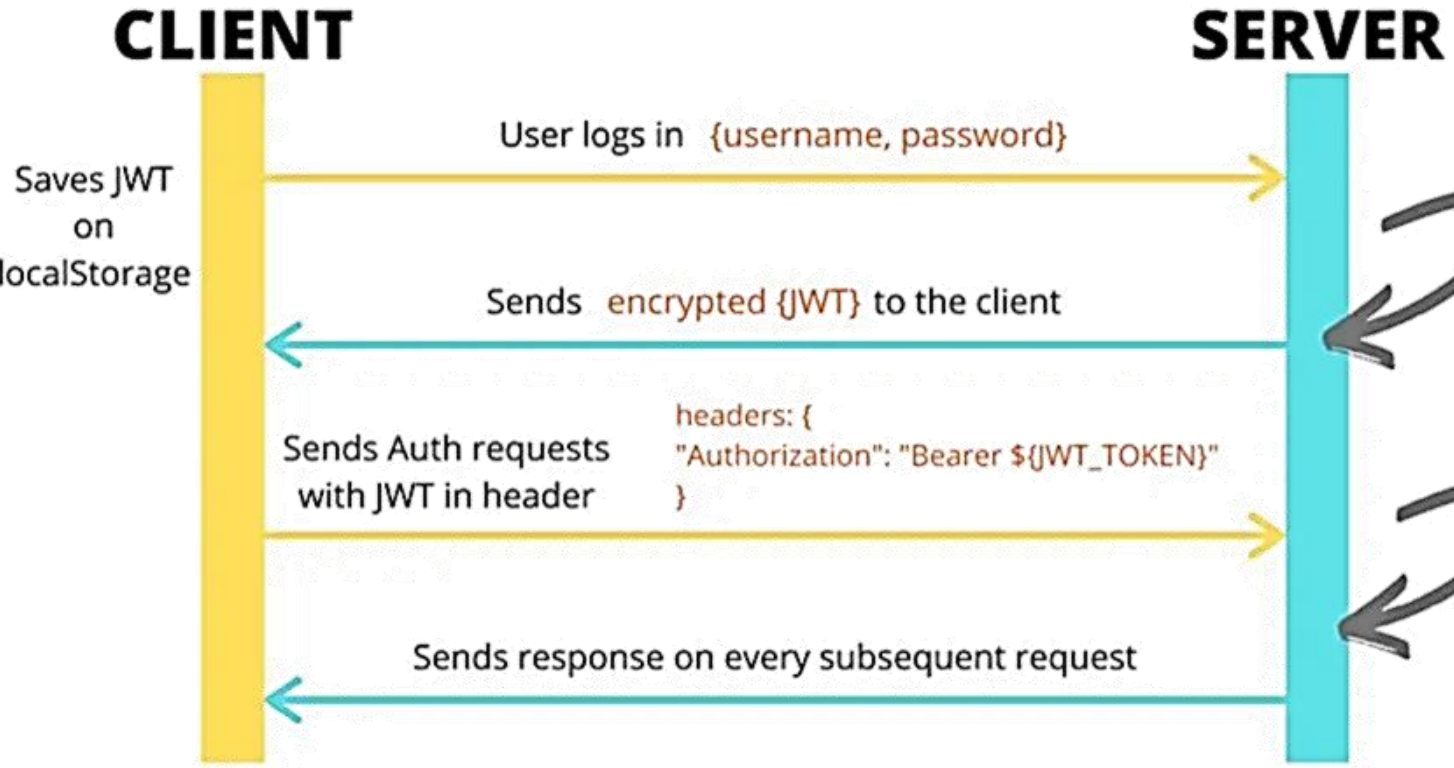


اجزای یک توکن JWT



- بخش Header
  - نوع توکن – JWT
  - الگوریتم رمز نگاری
- بخش Payload
  - اطلاعاتی در خصوص درخواست‌دهنده.
- بخش Signature
  - اطلاعاتی که با استفاده از آن‌ها می‌توان مطمئن شد که توکن در میانه‌ی مسیر تغییر نکرده‌است.

مراحل به کارگیری JWT



1. درخواست احراز هویت
2. تولید توکن و رمزنگاری آن
3. ارسال توکن به کلاینت
4. رد و بدل کردن توکن در جریات تعاملات بعدی.
5. درخواست کاربر در صورتی اجرا می‌شود که توکن در HTTP Header و به صورت صحیح موجود باشد.

پیاده‌سازی JWT در Spring Boot

```
<dependency>
  <groupId>io.jsonwebtoken</groupId>
  <artifactId>jjwt</artifactId>
  <version>0.9.1</version> <!-- Check for the latest version -->
</dependency>
```

Dependencies

- در ابتدا لازم است که تعدادی **Dependency** جدید نصب کنید.

```
<dependency>
  <groupId>javax.xml.bind</groupId>
  <artifactId>jaxb-api</artifactId>
  <version>2.3.1</version>
</dependency>
<dependency>
  <groupId>org.glassfish.jaxb</groupId>
  <artifactId>jaxb-runtime</artifactId>
  <version>2.3.1</version>
</dependency>
```

```
<dependency>
  <groupId>io.jsonwebtoken</groupId>
  <artifactId>jjwt-api</artifactId>
  <version>0.11.5</version>
</dependency>
<dependency>
  <groupId>io.jsonwebtoken</groupId>
  <artifactId>jjwt-jackson</artifactId>
  <version>0.11.5</version>
</dependency>
```

سرویس تولید توکن JWT

- برای تولید توکن و نیز برای لود کردن کاربر متناظر با هر توکن، نیازمند سرویس هستیم!
  - تولید توکن
  - نگاشت توکن
  - اعتبارسنجی توکن

```
@Component
public class JwtTokenService {

    3 usages
    private String SECRET_KEY = "your_secret_key";

    1 usage
    public String generateToken(String username) {...}

    // New method to extract and validate the token
    1 usage
    public String getUsernameFromRequest(HttpServletRequest request) {...}
```

به کارگیری سرویس JWT برای ساخت توکن

- برای این منظور، کنترلر مربوطه به این شکل تغییر می‌کند :

```
@PostMapping("/login")
public ResponseEntity<String> login(@RequestBody LoginRequest loginRequest, HttpServletRequest request) {
   try {
      Authentication authentication = authenticationManager.authenticate(
            new UsernamePasswordAuthenticationToken(loginRequest.getUserName(), loginRequest.getPassword()));
      SecurityContextHolder.getContext().setAuthentication(authentication);
      //return ResponseEntity.ok(request.getSession().getId());
      String token ="Bearer " + jwtTokenService.generateToken(loginRequest.getUserName());
      System.out.println(token);
      return ResponseEntity.ok().header(HttpHeaders.AUTHORIZATION,
            token).header(HttpHeaders.ACCESS_CONTROL_EXPOSE_HEADERS,
           "Authorization").build();
   } catch (AuthenticationException e) {
      return ResponseEntity.status(HttpStatus.UNAUTHORIZED).body("Login failed: " + e.getMessage());
   }
}
```

فیلتر جدید !

- همان طور که مشاهده شد، اقسام مختلفی از فیلترها وجود دارند که می‌توان آن‌ها را در Security Config گردآوری کرده و بهم وصل کرد.
- شما می‌توانید بسته به نیازتان، فیلتر جدیدی را بسازید و سپس در Security Config از آن استفاده کنید.
- یک فیلتر می‌تواند در مواقع مختلفی صدا زده شود : زمان فراخوانی سرولت، پس از فراخوانی سرولت و....
- ارزیابی توکن بایستی پس از هر درخواست اجرا شود و در نتیجه به فیلتر نیاز داریم!

https://www.baeldung.com/spring-onceperrequestfilter

فیلتر عمومی OncePerRequestFilter

- این یک فیلتر انتزاعی است که دارای متدهای مختلفی است که می‌توانید آن را پیاده سازی کنید.
- چه موقع فراخوانی میشه؟
  - پس از ارسال درخواست HTTP و دریافت آن ، این فیلتر اعمال می‌شود.

```
public class SimpleFilter extends OncePerRequestFilter {
    @Override
    protected void doFilterInternal(HttpServletRequest request, HttpServletResponse response, FilterChain filterChain) throws
ServletException, IOException {
        //Other Code
        filterChain.doFilter(request,response);
    }
}
```

فیلتر مجدد هر درخواست

```
protected void doFilterInternal(HttpServletRequest request, HttpServletResponse response, FilterChain filterChain)
throws ServletException, IOException {
   //Step 1 - get Authorization Header
   String jwtToken = request.getHeader(HttpHeaders.AUTHORIZATION);
   if(jwtToken!=null){
      //Step 2 - get Authenticated User
      String username = jwtTokenService.getUsernameFromRequest(request);

      //Step 3 - Setup Current User
      UserDetails customer = securityService.loadUserByUsername(username);

      //Step 4 Authenticate
      Authentication authentication =
            new UsernamePasswordAuthenticationToken(customer, null,customer.getAuthorities());
      SecurityContextHolder.getContext()
            .setAuthentication(authentication);
      System.out.println(SecurityContextHolder.getContext().getAuthentication());
   }
   //Step 5 - filter
   filterChain.doFilter(request,response);
}
```

استخراج Header مربوط به توکن

دریافت کاربر متناظر با توکن

استخراج کاربر

احراز هویت کاربر

فراخوانی فیلتر بعدی

کنترل دسترسی در سطح متد

کنترل دسترسی (Authorization)

```
@Configuration
@EnableWebSecurity
@EnableGlobalMethodSecurity(prePostEnabled = true)
public class SecurityConfig {

// other configurations

}
```

- تا کنون آموختیم که یکی از روش‌های اساسی برای کنترل سطح دسترسی استفاده از **Security Filter Chain** است.
- اگرچه این روش منجر به حفاظت **Endpoint** های برنامه می‌شود، اما سطح دسترسی را در موقع فراخوانی توابع برنامه بررسی نمی‌کند.
- برای آن که بتوان کنترل دسترسی را در سطح متدها برقرار کرد، بایستی در کلاس پیکره بندی امنیت، **method Security** را فعال کنیم.

احراز هویت با Auth2.0

فرآیند احراز هویت

- **احراز هویت:(Authentication)**

برنامه‌ی شخص ثالث (مثلاً اپلیکیشن ما) با ارسال درخواست برای دسترسی به منابع محافظت‌شده، فرآیند احراز هویت را آغاز می‌کند.

- **اجازه‌ی دسترسی:(Authorization)**

مالک منبع (کاربر) اجازه‌ی دسترسی به منابع خود را صادر می‌کند، که معمولاً از طریق ورود به سیستم انجام می‌شود.

- **ارسال کد مجوز:**

سرور مجوز (Authorization Server) کاربر را تأیید می‌کند و او را با یک **کد مجوز (authorization code)** به سمت کلاینت (اپلیکیشن ما) هدایت می‌کند.

فرآیند احراز هویت

- **دریافت توکن دسترسی:**

کلاینت با استفاده از کد مجوز، از سرور مجوز درخواست یک **توکن دسترسی (access token)** می‌کند. قالب این توکن در استاندارد مشخص نشده، اما معمولاً از JWT (JSON Web Token) استفاده می‌شود.

- **اعتبارسنجی توکن:**

سرور مجوز توکن را اعتبارسنجی می‌کند. اگر توکن معتبر باشد، به برنامه‌ی کلاینت اجازه استفاده از آن داده می‌شود.

- **دسترسی به منابع محافظت‌شده:**

کلاینت می‌تواند از این توکن برای دسترسی به منابع محافظت‌شده استفاده کند؛ مثلاً با فراخوانیendpoint های یک REST API.

تنظیمات گیت هاب