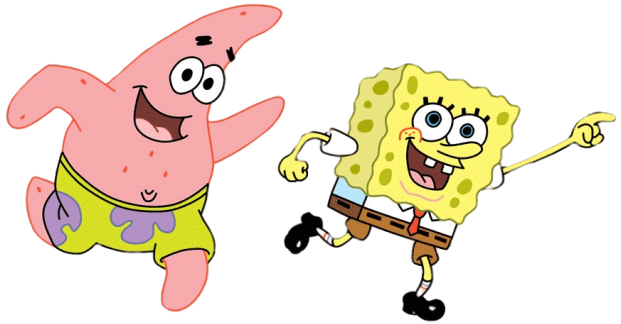
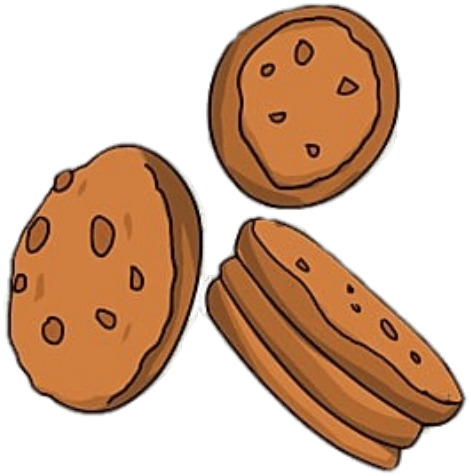
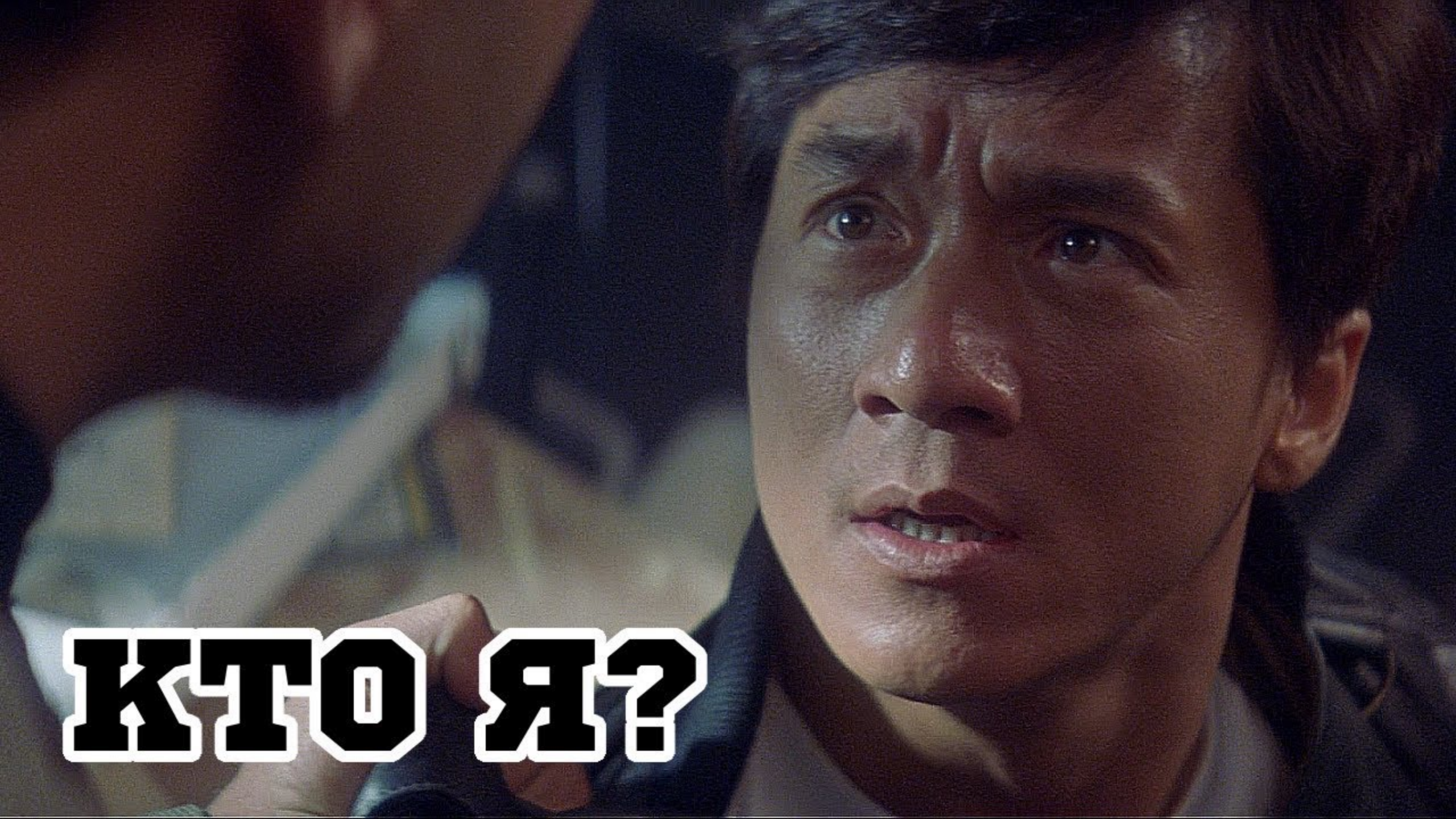




Spring Security

Повседневное и неочевидное

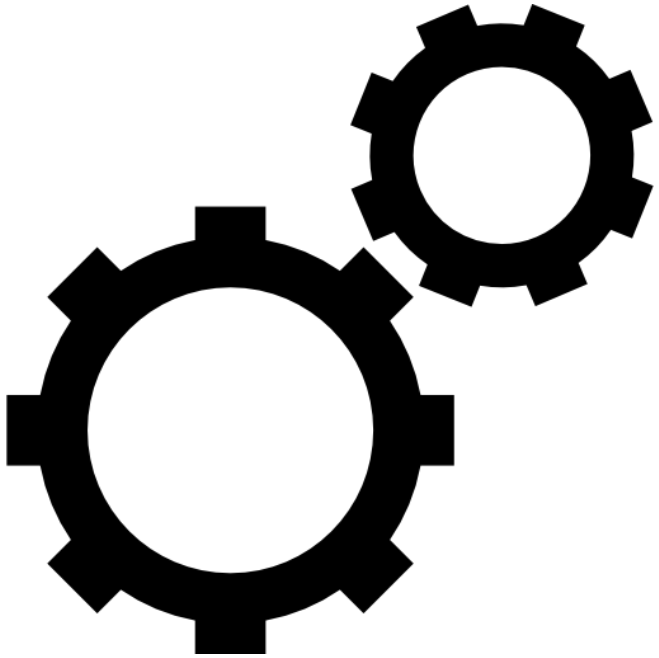

КТО Я?

Кислов Павел

- Работаю в компании Домклик.
- Около 6 лет в разработке.
- Люблю большой теннис
- Играю на нескольких музыкальных инструментах
- Катаюсь на серфе
- В свободное время преподаю Java.

О чем будем разговаривать?

+



Про что поговорим:

1) Нетривиальные конфиги и настройки.

2) Штуки, знанием которых, мы, как backend-разработчики пренебрегаем.

3) Какой Spring Security имеет вид в актуальной версии?

4) Как на него переехать, что поменялось и что нового?

Наше умение пользоваться Spring Security:

Решив написать домашний проект,
где все приходится делать с нуля
(Визжим)

На работе, где, чаще всего,
все было налажено до нас.
(Жонглируем гирями и делаем финты ушами)

Может ли что-то пойти не так и есть ли уязвимости?

```
override fun configure(http: HttpSecurity) {
    http
        .csrf().disable()
        .addFilterBefore(JWTTokenFilter(jwtTokenService), BasicAuthenticationFilter::class.java)
        .authorizeRequests()
        .antMatchers("/registration").permitAll()
        .mvcMatchers("/clients/all").hasRole(Role.MANAGER.name)
        .mvcMatchers("/clients/{id}").hasAnyRole(Role.MANAGER.name, Role.DOCTOR.name)
        .mvcMatchers("/visits/doctor/{doctorId: regex}").hasAnyRole(Role.MANAGER.name)
        .mvcMatchers("/employees/doctors").permitAll()
        .mvcMatchers("/auth/jwt/authenticate").permitAll()
        .mvcMatchers("/clients").hasAnyRole(Role.MANAGER.name, Role.DOCTOR.name)
        .anyRequest().authenticated()
}
```

```
http
    .csrf().disable()
    .addFilterBefore(JWTTokenFilter(jwtTokenService), BasicAuthenticationFilter::class.java)
    .authorizeRequests()
```

1. Использование CSRF-токенов выключено, а session management работает.

2. JWTTokenFilter добавлен перед BasicAuthenticationFilter, а не вместо но HttpBasic не отключен.

3. JWT и HTTP Basic в ондном конфиге. Мы смешали в одном месте механизмы с сессиями и без.

Какими знаниями мы, как бекенд разработчики чаще всего перенебрегаем?

1. Куки - Это что-то фронтендерское. Пусть фронтендеры и разбираются.

2. Сессии - Запросы приходят с фронта. Пусть фронтендеры и разбираются.

3. Процессы аутентификации/идентификации/ авторизации, способы и их совместимость.
- Да некогда мне/ я тут важным занят/ я больше про архитектуру, алгоритмы, пиво и сухарики.

Давайте разбираться.

SPRING BOOT – ЭТО ВСЕ, ЧТО НАМ НУЖНО СЕГОДНЯ
(С)ДЖЕЙСОН СТЕТХЕМ

spring

spring boot

Или любой другой контейнер сервлетов

Tomcat embeded

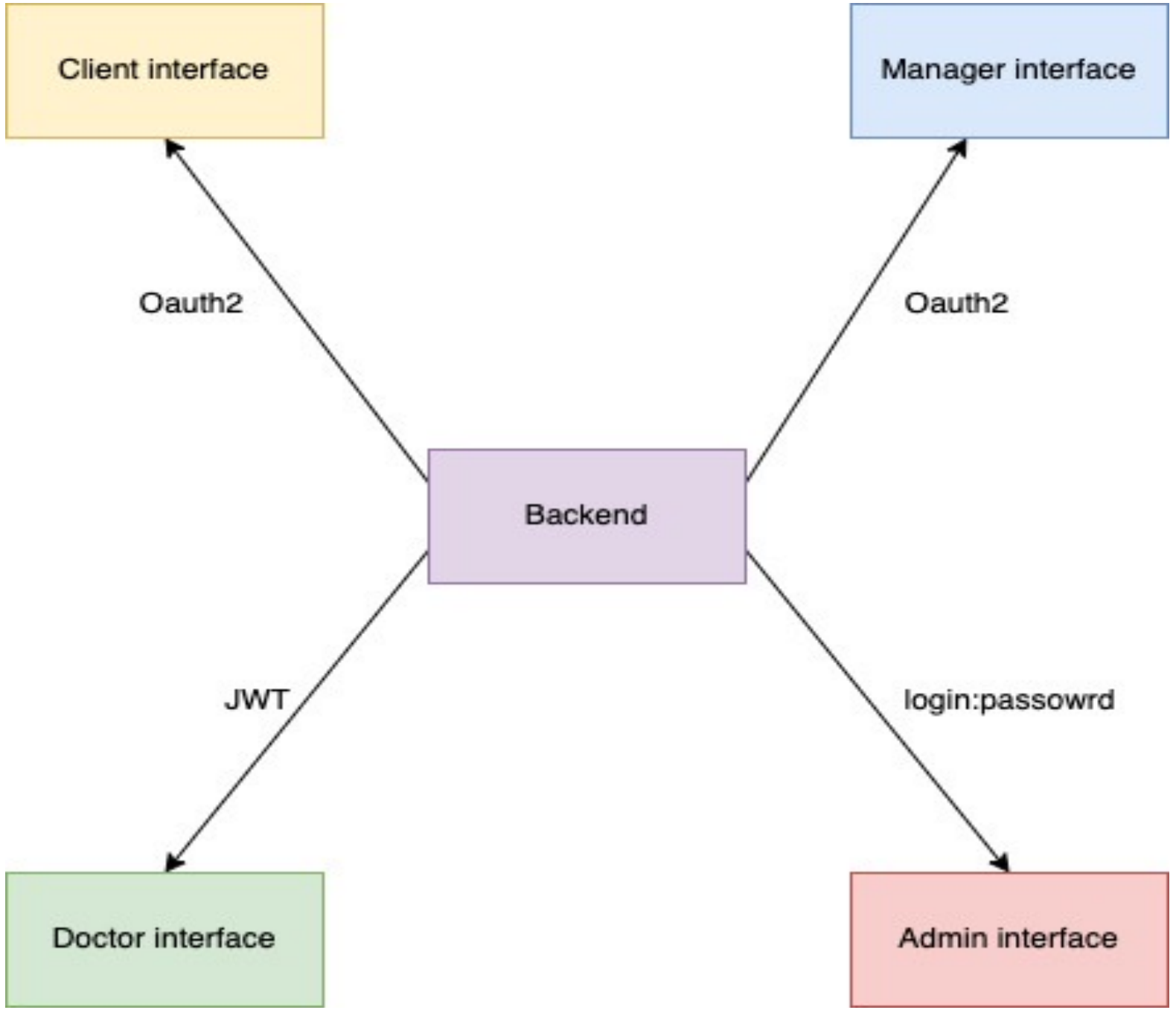
spring boot

autoconfiguration

Какую идеологию нам предлагает «Spring boot friendly design»?

- Ключевые инфраструктурные элементы нашего приложения стоит отдать под управление Spring
- Они по возможности бины
- Для них есть конфигурации
- Для всего что мы используем и не настраиваем руками есть автоконфигурации и дефолтные конфиги
- Все, что делает наше приложение на старте – это создает бины и выполняет наши конфиги в определенном порядке


[ЧЕЛОКЛИНИКА]


Client interface
Manager interface
Oauth2
Oauth2
Backend
JWT
login:passowrd
Doctor interface
Admin interface

Возникающие вопросы

1. Как запрос доходит до контроллера и как после его работы возвращается и формируется ответ?
2. Я пишу свой конфиг и для работы со Spring Security приходится наследовать его от WebSecurityConfigurerAdapter и имплементить какие-то методы, а потом работать с httpSecurity и WebSecurity. Что это все такое? Зачем оно мне ? Я же жил нормально!!!
3. Роли и привилегии - в чем разница?
4. Как мы храним роли и привилегии и храним ли вообще?
5. Нужны ли нам свои фильтры для хождения по логину и паролю?
6. В каких вообще случаях нам нужны свои фильтры?
7. Какие они вообще есть и зачем их столько?
8. Какой вообще у фильтров порядок выполнения?
9. Аутентификация и авторизация – в чем разница и как они работают в spring security?
10. Несколько провайдеров аутентификации - как и зачем?
11. RequestMatchers – зачем их 3 типа и в чем разница?
12. Security Filters vs Global method security – что-то нашел в курсах индусов и на стек оверфлоу. В чем разница подходов и зачем это все?
13. Несколько Security-конфигов одновременно – как это работает?
14. В большинстве видео гайдов выключают защиту CSRF, но говорят этого не делать в реальных проектах. Но выключают. Что это такое и как работает?
15. Я написал свой проект и хочу мигрировать на Spring Boot 3. Что мне нужно знать для миграции в части Security?
16. Как меняется использование спринг секьюрити, если у меня монолит/микросервисная архитектура?

Наш типичный конфиг

```
@Configuration
class SecurityConfig : WebSecurityConfigurerAdapter() {

    override fun configure(http: HttpSecurity?) {
        super.configure(http)
        // какие-то настройки
    }

    override fun configure(web: WebSecurity?) {
        super.configure(web)
        // какие-то настройки
    }
}
```

Наш типичный конфиг

```
@Configuration
class SecurityConfig : WebSecurityConfigurerAdapter() {

    override fun configure(http: HttpSecurity?) {
        super.configure(http)
        // какие-то настройки
    }

    override fun configure(web: WebSecurity?) {
        super.configure(web)
        // какие-то настройки
    }
}
```

Начинаем разбираться:

WebSecurity – это класс, который используется для настройки глобальной политики безопасности и унифицированных настроек, которые будут применены ко всем цепочкам фильтров.

HttpSecurtity – это класс, который используется для настройки конкретных цепочек фильтров.

FilterChainProxy – это фильтр-делегат, который в последствии определяет, какие конкретные FilterChain должны быть вызваны. По факту, этот фильтр производит первичный отсев запросов, а последующую обработку передает далее.

WebSecurityConfigurerAdapter – это просто класс для того, чтобы настраивать нашу безопасность.

Как это все работает:

1. В spring security существует класс WebSecurityConfiguration

2. В нем есть метод setFilterChainProxySecurityConfigurer

3. В нем получаются все конфигурации (созданные пользователем(наш SecurityConfiguration) и дефолтные)

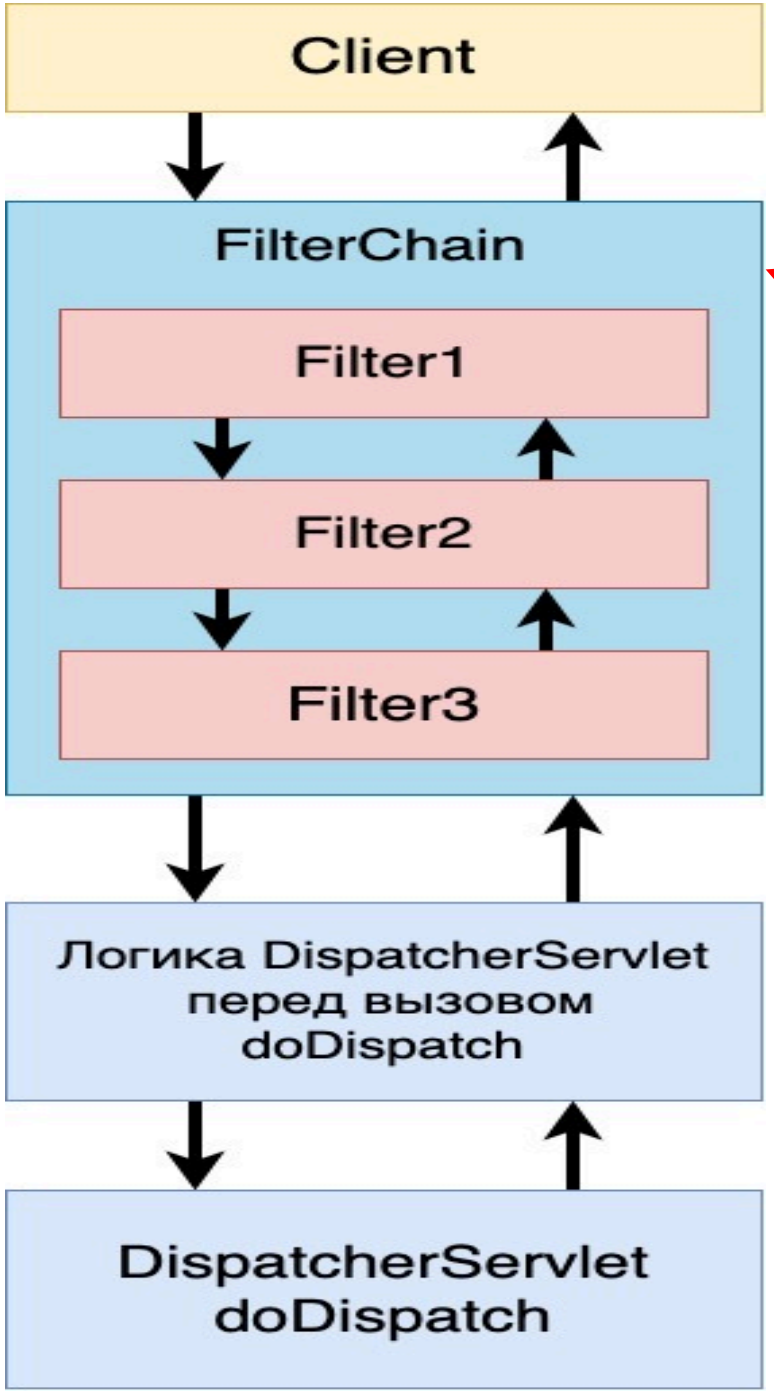
4. Далее создается объект WebSecurity

5. В него кладутся все конфигурации

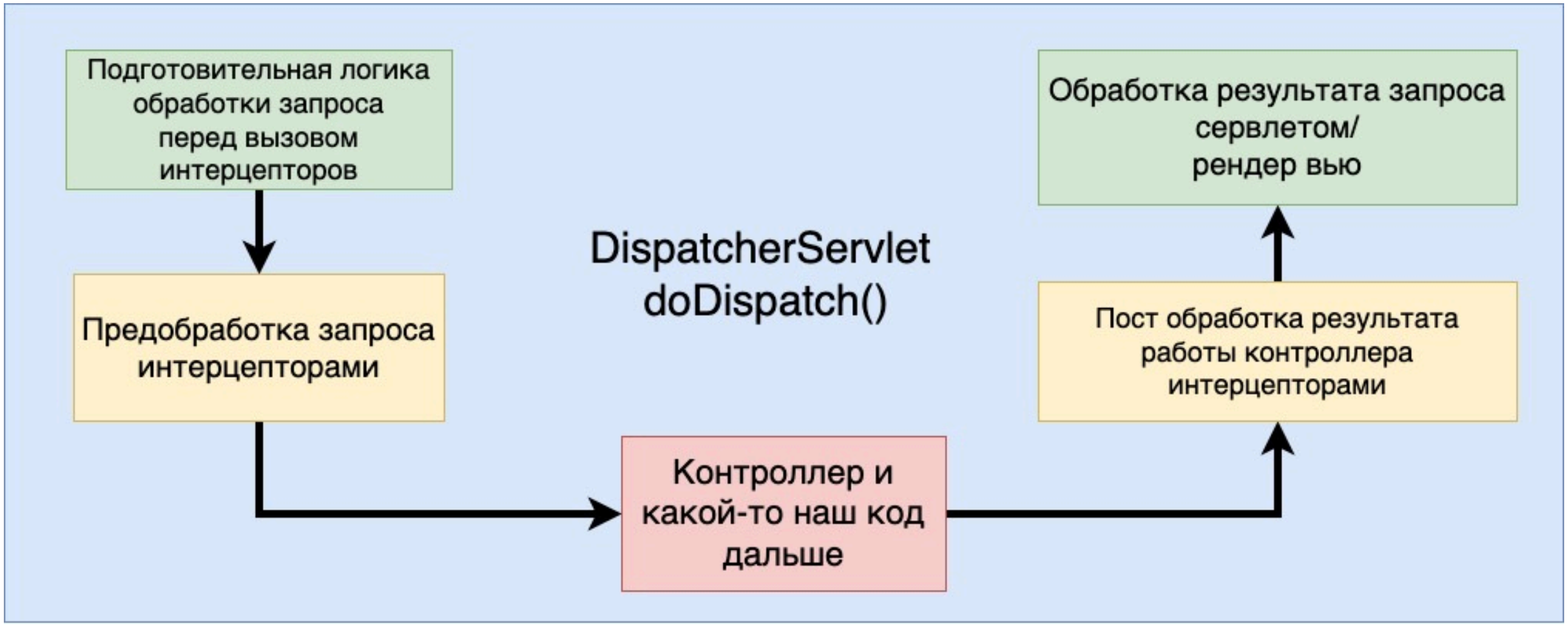
6. Затем вызывается метод build у WebSecurity.

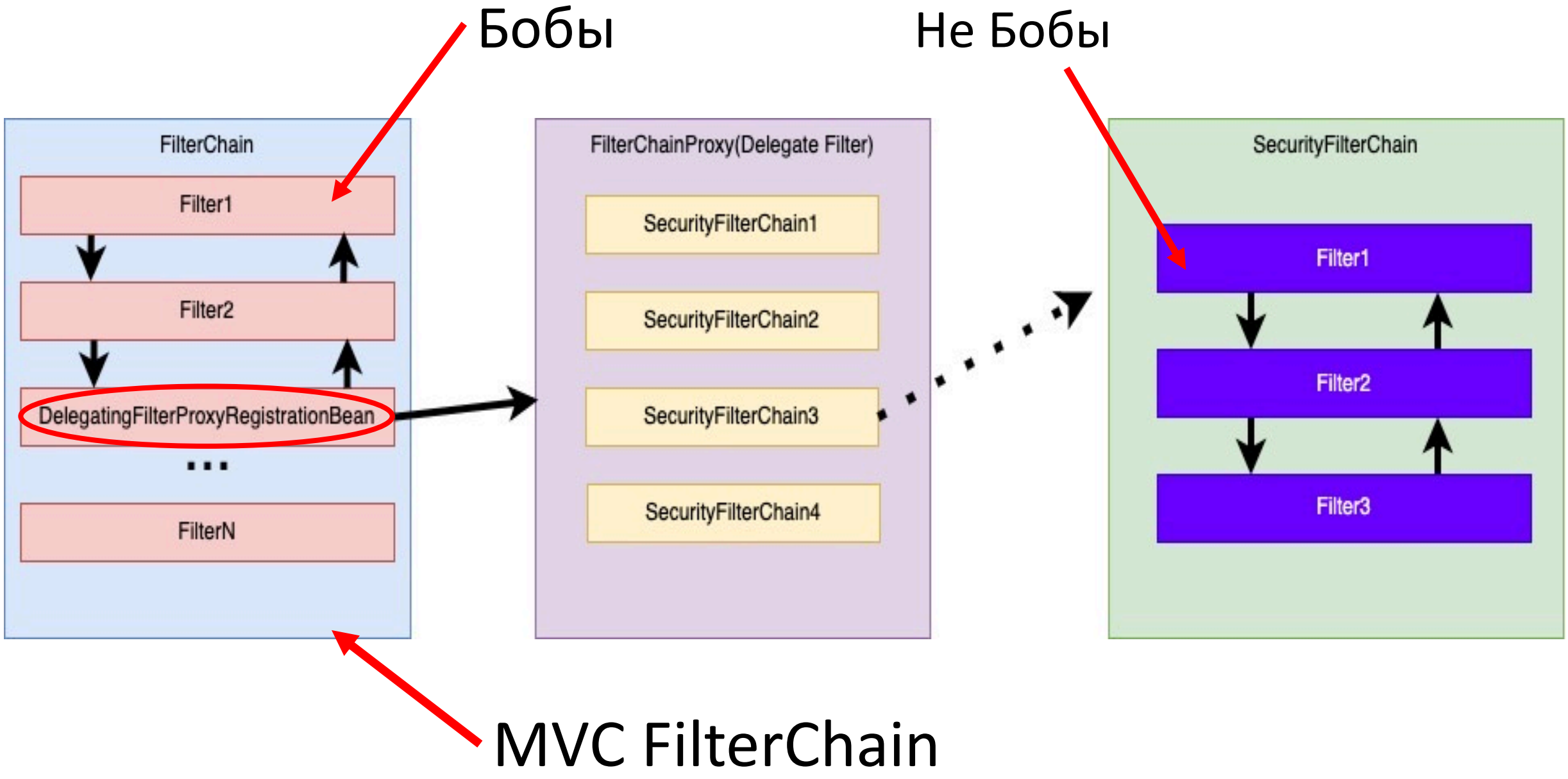
7. В нем для каждой конфигурации вызывается метод configure(WebSecurity)

Какой путь проходит запрос в нашем приложении?



Servlet FilterChain


Подготовительная логика обработки запроса перед вызовом интерцепторов
Предобработка запроса интерцепторами
DispatcherServlet
doDispatch()
Контроллер и какой-то наш код дальше
Пост обработка результата работы контроллера интерцепторами
Обработка результата запроса сервлетом/ рендер вью


Бобы
Не Бобы
FilterChain
Filter1
Filter2
DelegatingFilterProxyRegistrationBean
...
FilterN
FilterChainProxy(Delegate Filter)
SecurityFilterChain1
SecurityFilterChain2
SecurityFilterChain3
SecurityFilterChain4
SecurityFilterChain
Filter1
Filter2
Filter3
MVC FilterChain

Аутентификация/Идентификация/Авторизация

Идентификация — процесс, в результате выполнения которого для субъекта идентификации выявляется его идентификатор, однозначно определяющий этого субъекта в информационной системе. Мы понимаем кто перед нами.

Аутентификация — процедура проверки подлинности, например проверка подлинности пользователя путем сравнения введенного пароля с паролем из базы данных.

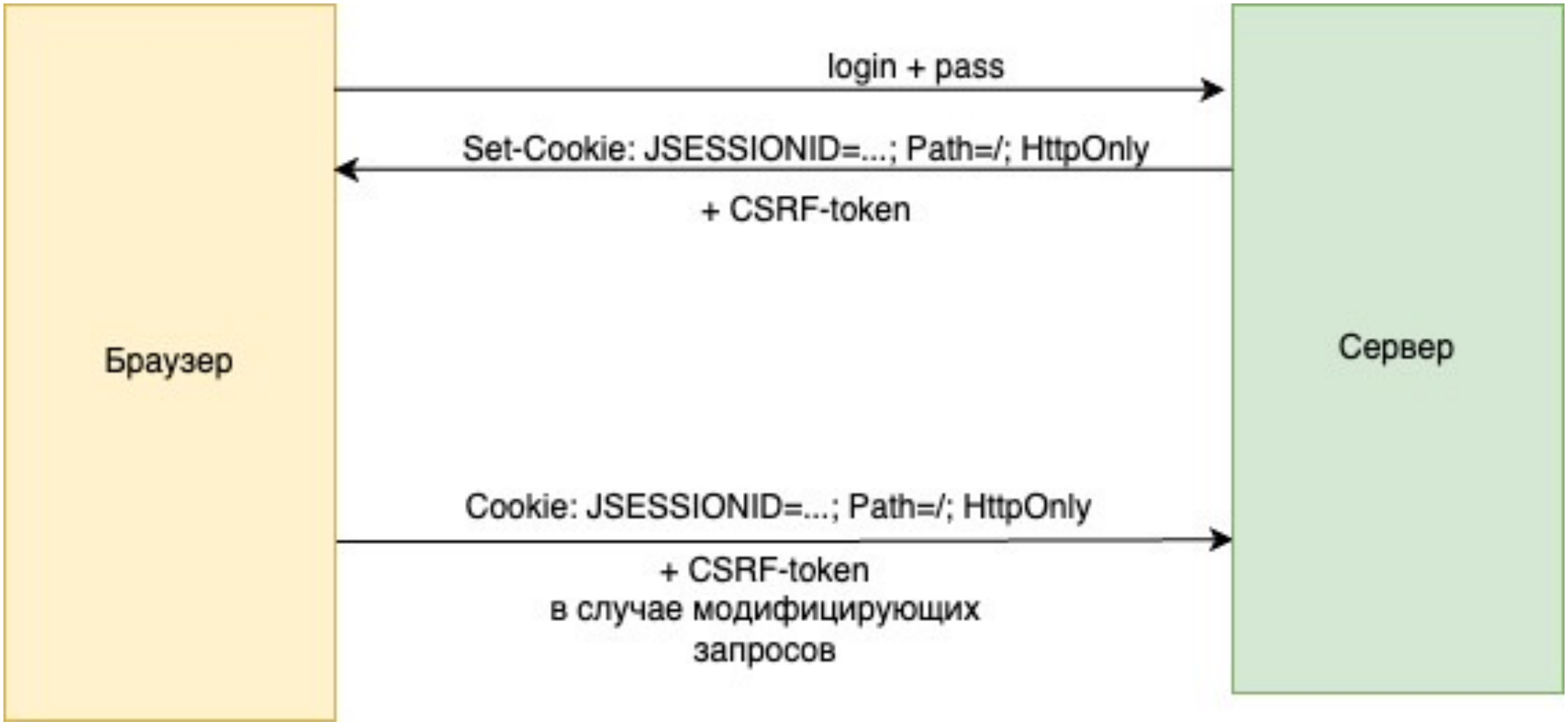
Авторизация — предоставление определенному лицу или группе лиц прав на выполнение действий. Например, можно ли пользователю выполнять тот или метод контроллера.

Сессии/Куки/Токены

Cookie – какие-то данные, которые наш браузер хранит нашей машине и отдает на сервер. Отличаются по времени хранения. Бывают те, которые валидны долго, бывают часто обновляемые. Для Spring – это просто заголовок, откуда можно взять полезное.

Http session – некоторый отрезок времени, в рамках которого сервер может принимать запросы от клиента, не требуя повторного прохождения идентификации и аутентификации.

Токен – некоторая последовательность символов, необходимая пользователю для процессов идентификации/авторизации. Может содержать в себе зашифрованные данные или являться случайным набором символов без скрытого содержания(blind-token).


Браузер
login + pass
Set-Cookie: JSESSIONID=...; Path=/; HttpOnly
+ CSRF-token
Cookie: JSESSIONID=...; Path=/; HttpOnly
+ CSRF-token
в случае модифицирующих
запросов
Сервер

Сессия

Не нужна	Нужна
Oauth2	login:password
JWT	

Почему сессия не нужна при использовании JWT и OAUTH 2?

JWT

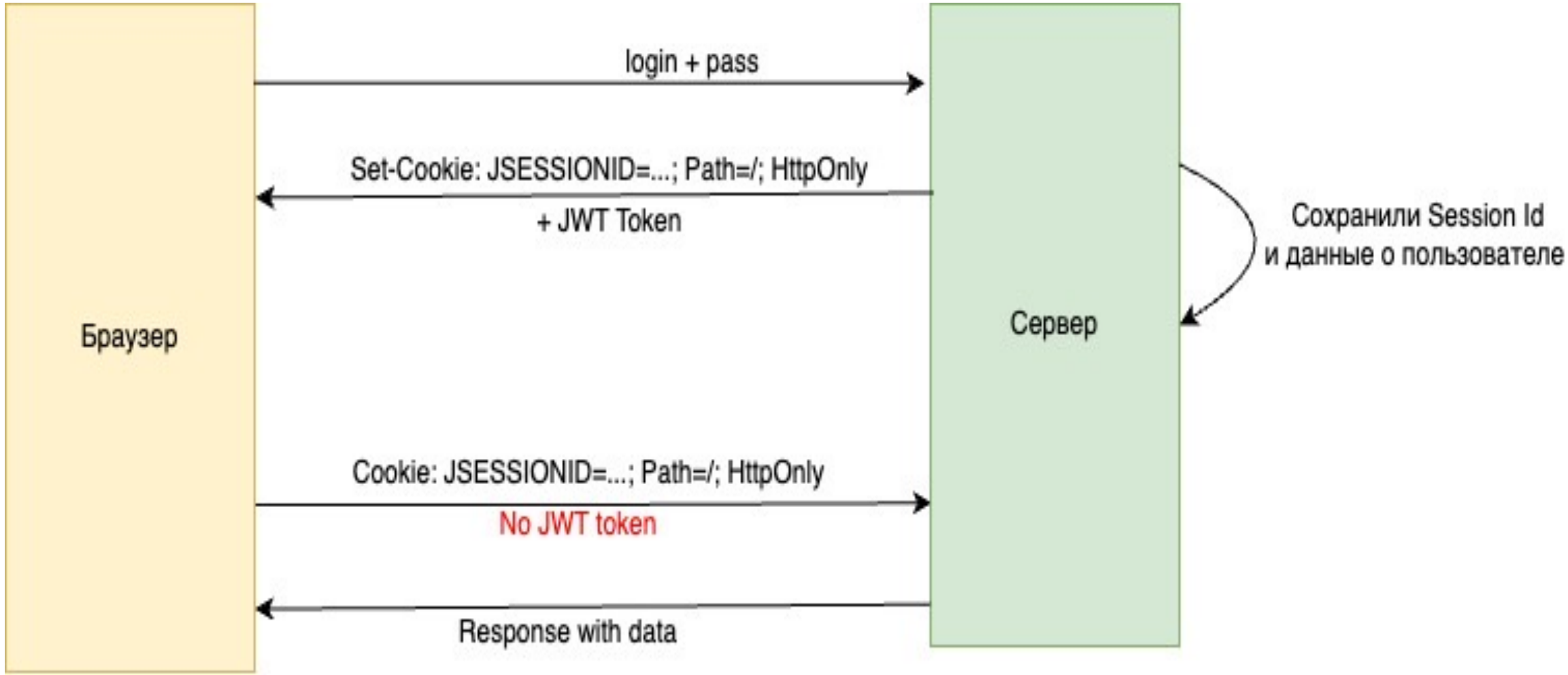
1. Серверу не обязательно хранить данные о сеансе пользователя у себя, так как JWT токен уже содержит в себе необходимую информацию.
2. Часто участвующих в процессе работы с токеном больше чем двое. Чем в большем количестве мест вы храните данные пользователя, тем больше шанс, что они уязвимы.

OAUTH2

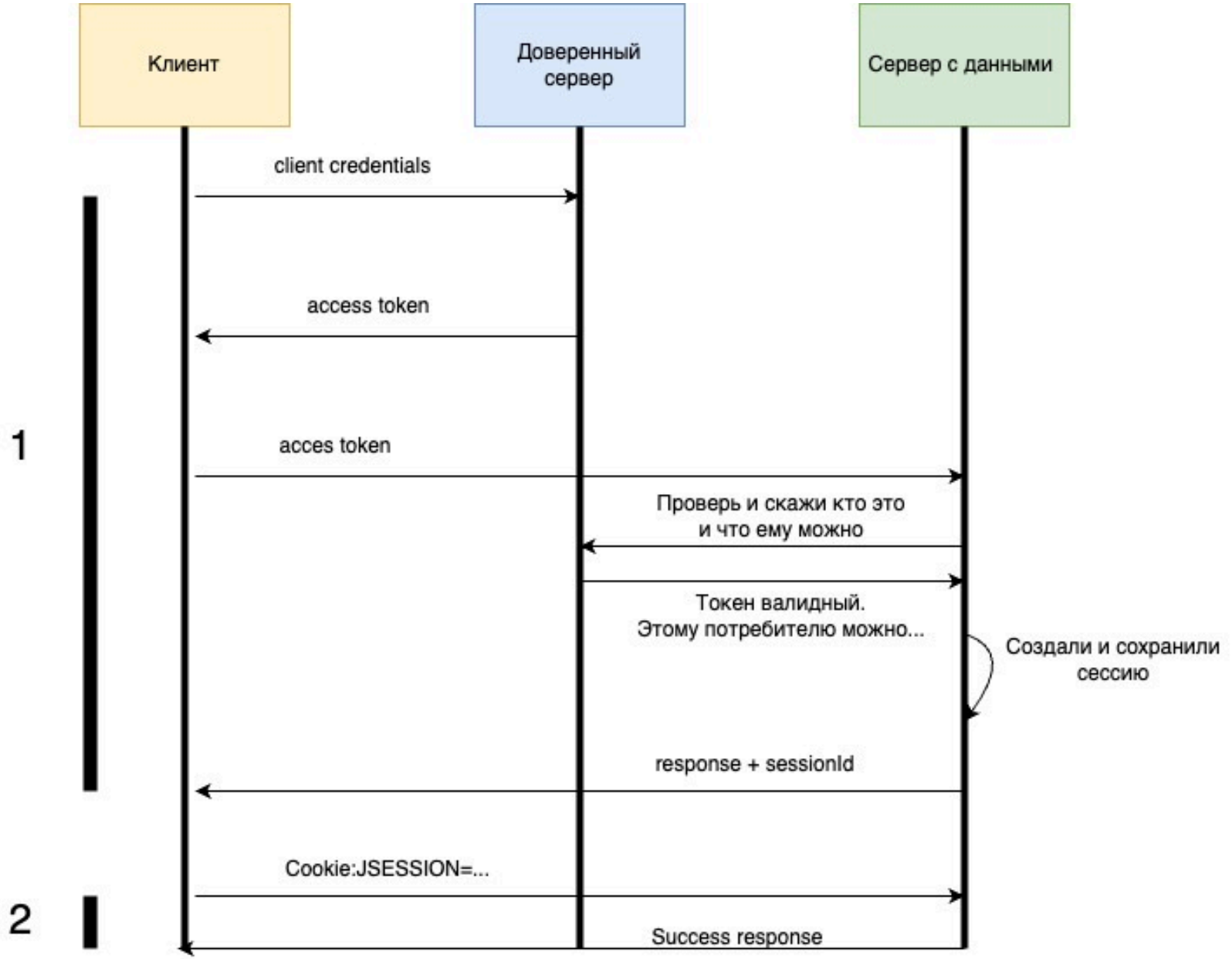
1. При использовании OAUTH2 часто применяется механизм JWT-токенов и, значит, токен уже содержит в себе все необходимые данные.
2. При использовании OAUTH2 с blind-token сверка токена после получения происходит с доверенным приложением и мы не хотели бы, чтобы эта процедура пропускалась.

Сверка токена должна происходить при каждом запросе

Пример уязвимости с JWT:



Пример уязвимости с OAUTH2 и blind-токеном:



Для самостоятельного изучения:

Про CSRF

Про XSS

Про XXE

ДОКЛАД Security XML 13.10 / 16:15 — 17:00 UTC+3 (Europe/Moscow) ЗАЛ 3

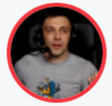
Как обработка XML приводит к проблемам с безопасностью? Разбираемся с XXE

Обработка XML может приводить к неожиданным проблемам с безопасностью приложений. Например, к утечкам данных. Как? Этому и будет посвящён доклад.

Основная тема — уязвимость XXE. Поговорим о том, из-за чего вообще при работе с XML возникают дефекты безопасности. Разберёмся с тем, какие виды XXE бывают и в чём их особенности. Конечно, затронем вопросы атаки и защиты. Для лучшего понимания материала спикер продемонстрирует примеры реальных уязвимостей из open source-проектов.

RU

Спикеры



Сергей Васильев

Биография Страница спикера →

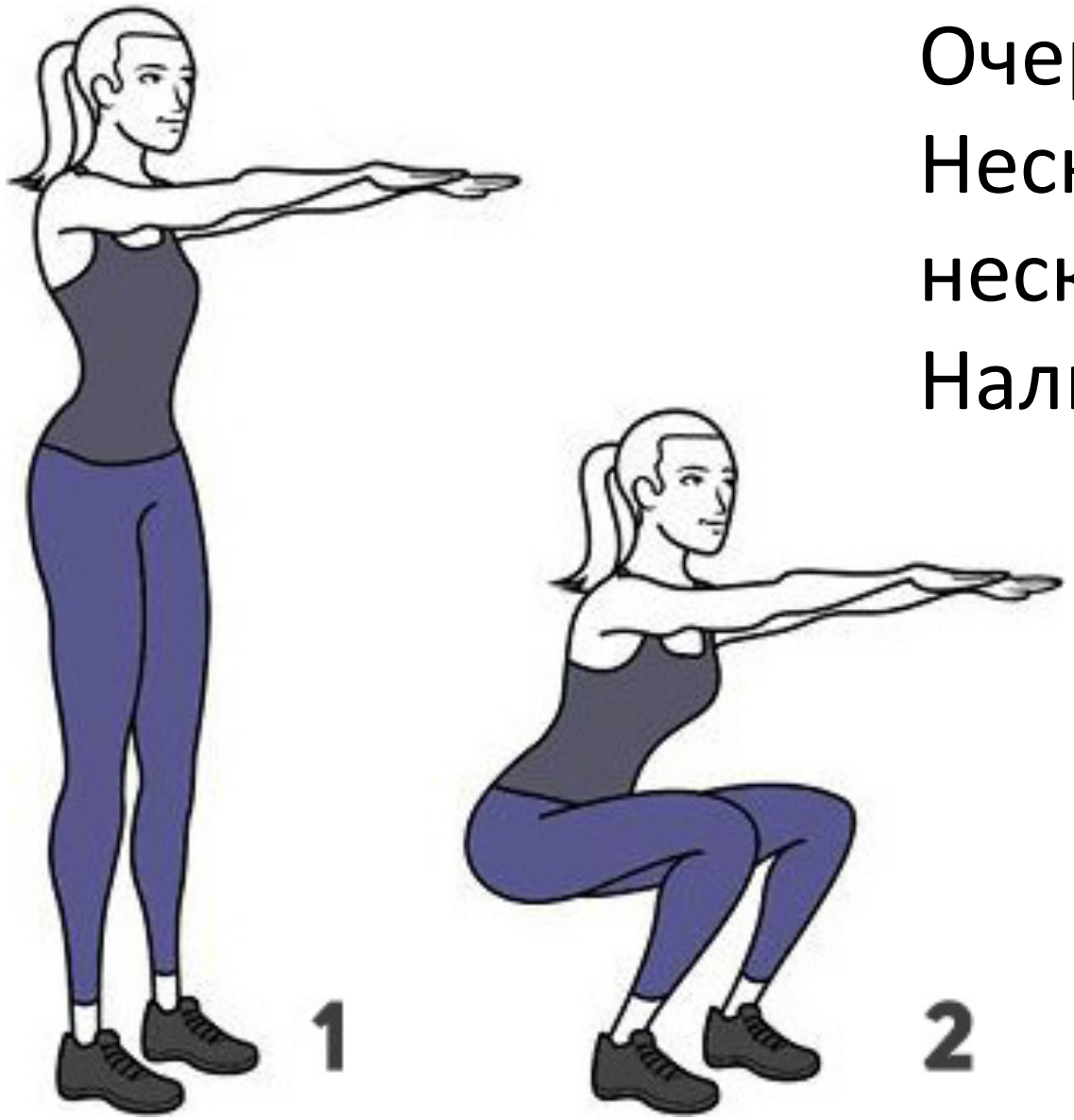
Эксперты

Андрей Когунь

КРОК

Биография Страница эксперта →

Ссылка на доклад



Очередное приседание:
Несколько конфигов,
несколько цепочек фильтров.
Наличие сессий и их отсутствие.

Все фильтры Spring Security:

- ForceEagerSessionCreationFilter
- ChannelProcessingFilter
- WebAsyncManagerIntegrationFilter
- SecurityContextPersistenceFilter
- HeaderWriterFilter
- CorsFilter
- CsrfFilter
- LogoutFilter
- OAuth2AuthorizationRequestRedirectFilter
- Saml2WebSsoAuthenticationRequestFilter
- X509AuthenticationFilter
- AbstractPreAuthenticatedProcessingFilter
- CasAuthenticationFilter
- OAuth2LoginAuthenticationFilter
- Saml2WebSsoAuthenticationFilter
- UsernamePasswordAuthenticationFilter
- DefaultLoginPageGeneratingFilter
- DefaultLogoutPageGeneratingFilter
- ConcurrentSessionFilter
- DigestAuthenticationFilter
- BearerTokenAuthenticationFilter
- BasicAuthenticationFilter
- RequestCacheAwareFilter
- SecurityContextHolderAwareRequestFilter
- JaasApiIntegrationFilter
- RememberMeAuthenticationFilter
- AnonymousAuthenticationFilter
- OAuth2AuthorizationCodeGrantFilter
- SessionManagementFilter
- ExceptionTranslationFilter
- AuthorizationFilter
- SwitchUserFilter

Цепочка фильтров по-умолчанию

- WebAsyncManagerIntegrationFilter
- SecurityContextPersistenceFilter
- HeaderWriterFilter
- CsrfFilter
- LogoutFilter
- BasicAuthenticationFilter
- RequestCacheAwareFilter
- SecurityContextHolderAwareRequestFilter
- AnonymousAuthenticationFilter
- SessionManagementFilter
- ExceptionTranslationFilter
- FilterSecurityInterceptor

SecurityContextPersistenceFilter – если в репозитории севвий есть данные о сессии, то достанет их и заполнит SecurityContext на основании имеющихся данных. Умеет сохранить SecurityContext в репозитории сам.

SessionManagementFilter – выполняет действия связанные с сессией. Подменяет JSESSIONID(идентификатор сессии), регулирует количество параллельных сессий. Содержит внутри несколько делегатов-стратегий для работы с сессиями.
По умолчанию 2: ChangeSessionIdAuthenticationStrategy и CsrfAuthenticationStrategy. Подменяется JSESSIONID после логина и, если передан CSRF, то генерится новый.

AuthenticationFilter – аутентифицирует пользователя, помещает в SecurityContextHolder полученный SecurityContext.

HeaderWriterFilter – просто фильтр, заполняющий заголовки для ответов. Может быть полезно добавить определенные заголовки, которые включают защиту браузера. Например, X-Frame-Options, X-XSS-Protection и X-Content-Type-Options.

CsrfFilter - применяет защиту от CSRF. Следует помнить, что CsrfFilter вызывается для любого запроса, позволяющего изменить состояние. Еще, это означает, что необходимо убедиться, что веб-приложение следует правильной семантике REST (т. е. не изменяет состояние с помощью HTTP-методов GET, HEAD, TRACE, OPTIONS).

LogoutFilter – разлогинивает пользователя. Прогоняет набор логаут-хендлеров, публикует ивент об успешном «разлогине» и редиректит на указанный/дефолтный урл.

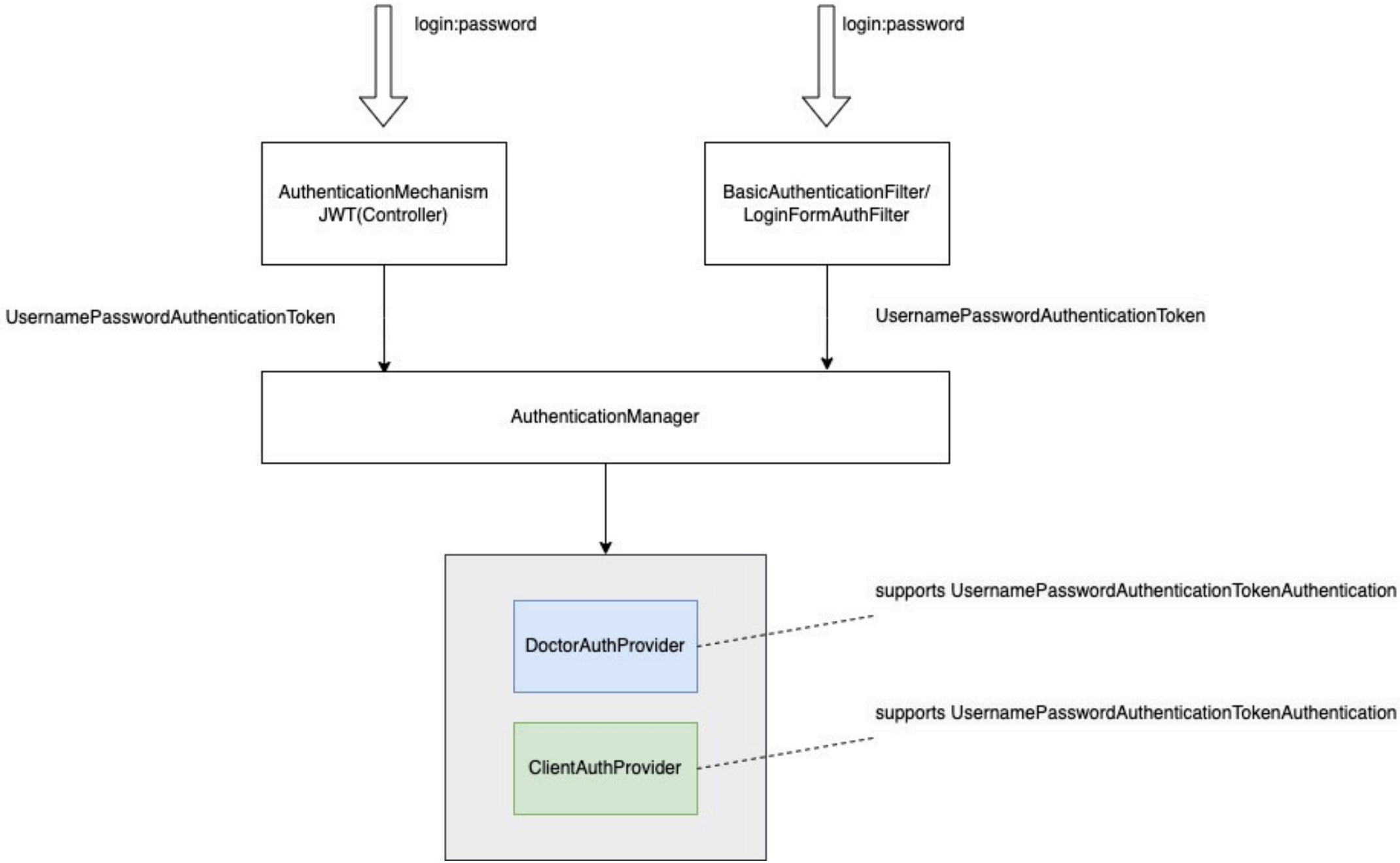
RequestCacheAwareFilter – отвечает за воссоздание сохраненного запроса, если он кэширован и соответствует текущему запросу. (Вы заходите куда-то не аутентифицировавшись, вас редиректит на страницу входа, вы входите и после вас перекидывает на ту, которую запрашивали изначально. Именно этот фильтр за это отвечает)

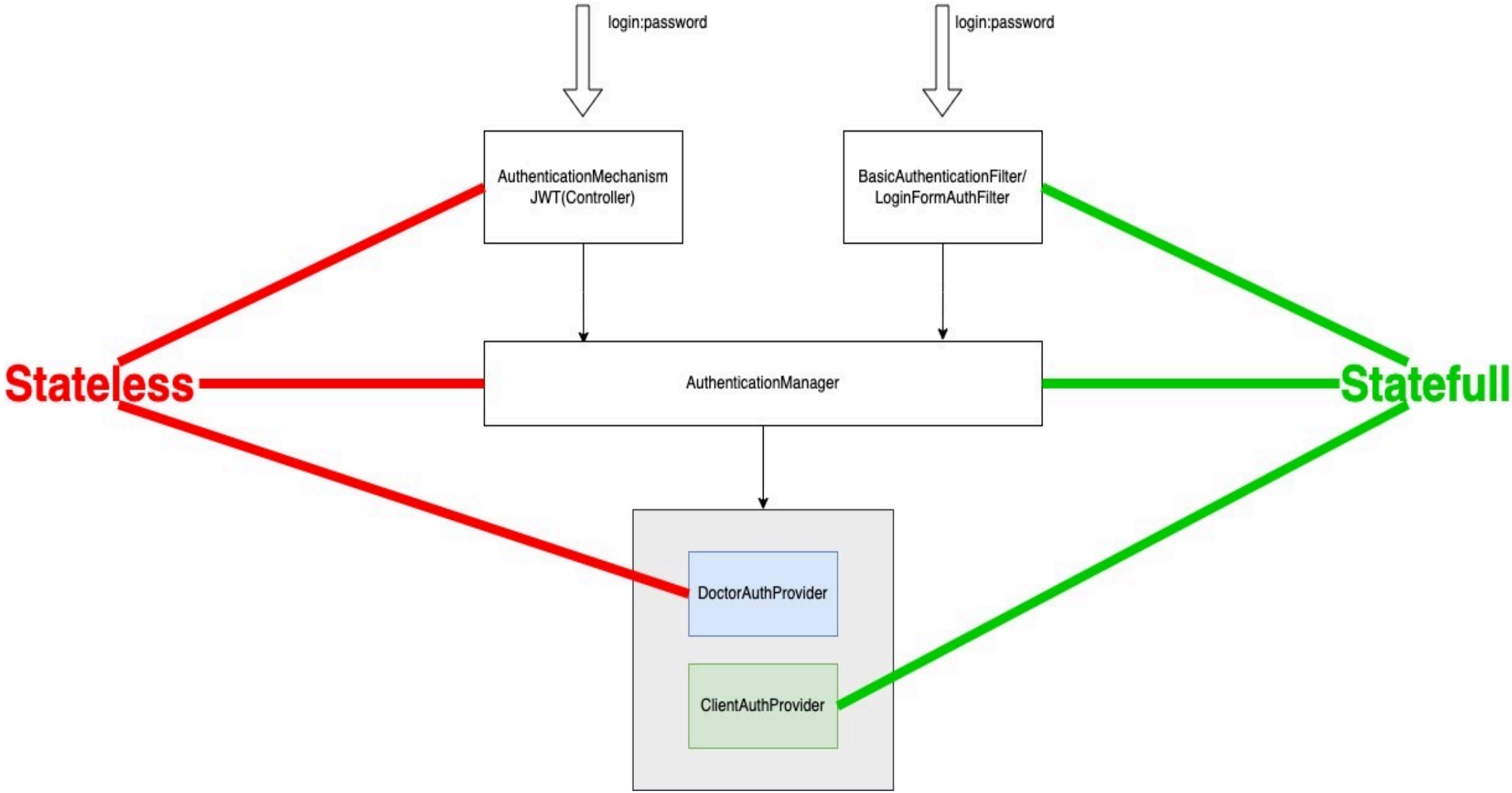
SecurityContextHolderAwareRequestFilter - Фильтр, который оборачивает ServletRequest в SecurityContextHolderAwareRequestWrapper, который реализует методы безопасности API сервлета.

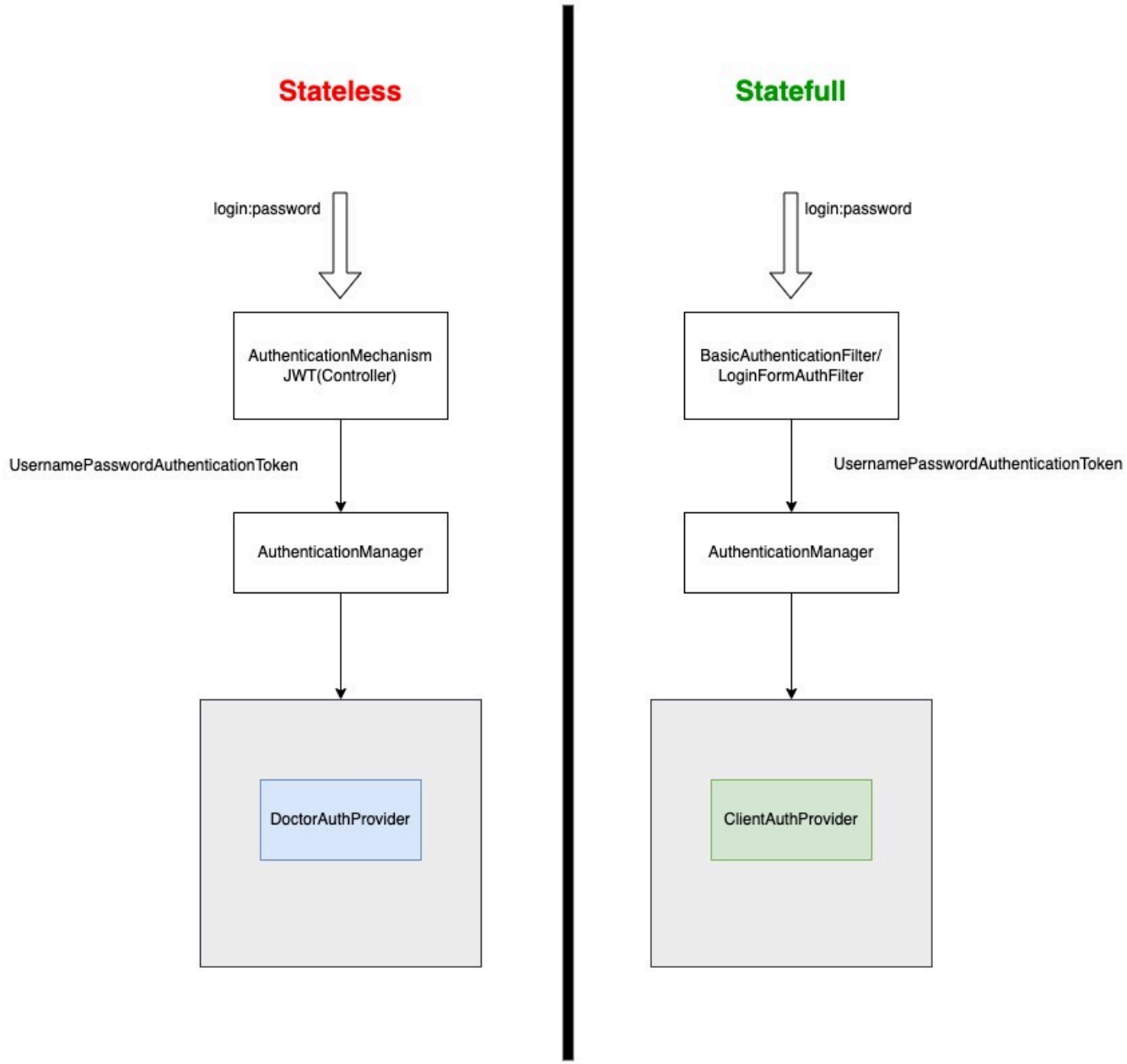
ExceptionTranslationFilter – фильтр, который трансформирует возникающие исключения(в том числе относящиеся к Security) в HttpResponse с определенными кодами.

FilterSecurityInterceptor – выполняется на последних этапах. Авторизует запрос пользователя и решает имеет ли он доступ к запрашиваемому ресурсу.

Подробно и просто про фильтры


login:password
login:password
AuthenticationMechanism
JWT(Controller)
BasicAuthenticationFilter/
LoginFormAuthFilter
UsernamePasswordAuthenticationToken
UsernamePasswordAuthenticationToken
AuthenticationManager
DoctorAuthProvider
ClientAuthProvider
supports UsernamePasswordAuthenticationTokenAuthentication
supports UsernamePasswordAuthenticationTokenAuthentication


login:password
login:password
AuthenticationMechanism
JWT(Controller)
BasicAuthenticationFilter/
LoginFormAuthFilter
Stateless
AuthenticationManager
Statefull
DoctorAuthProvider
ClientAuthProvider


Stateless
login:password
AuthenticationMechanism
JWT(Controller)
UsernamePasswordAuthenticationToken
AuthenticationManager
DoctorAuthProvider
Statefull
login:password
BasicAuthenticationFilter/
LoginFormAuthFilter
UsernamePasswordAuthenticationToken
AuthenticationManager
ClientAuthProvider

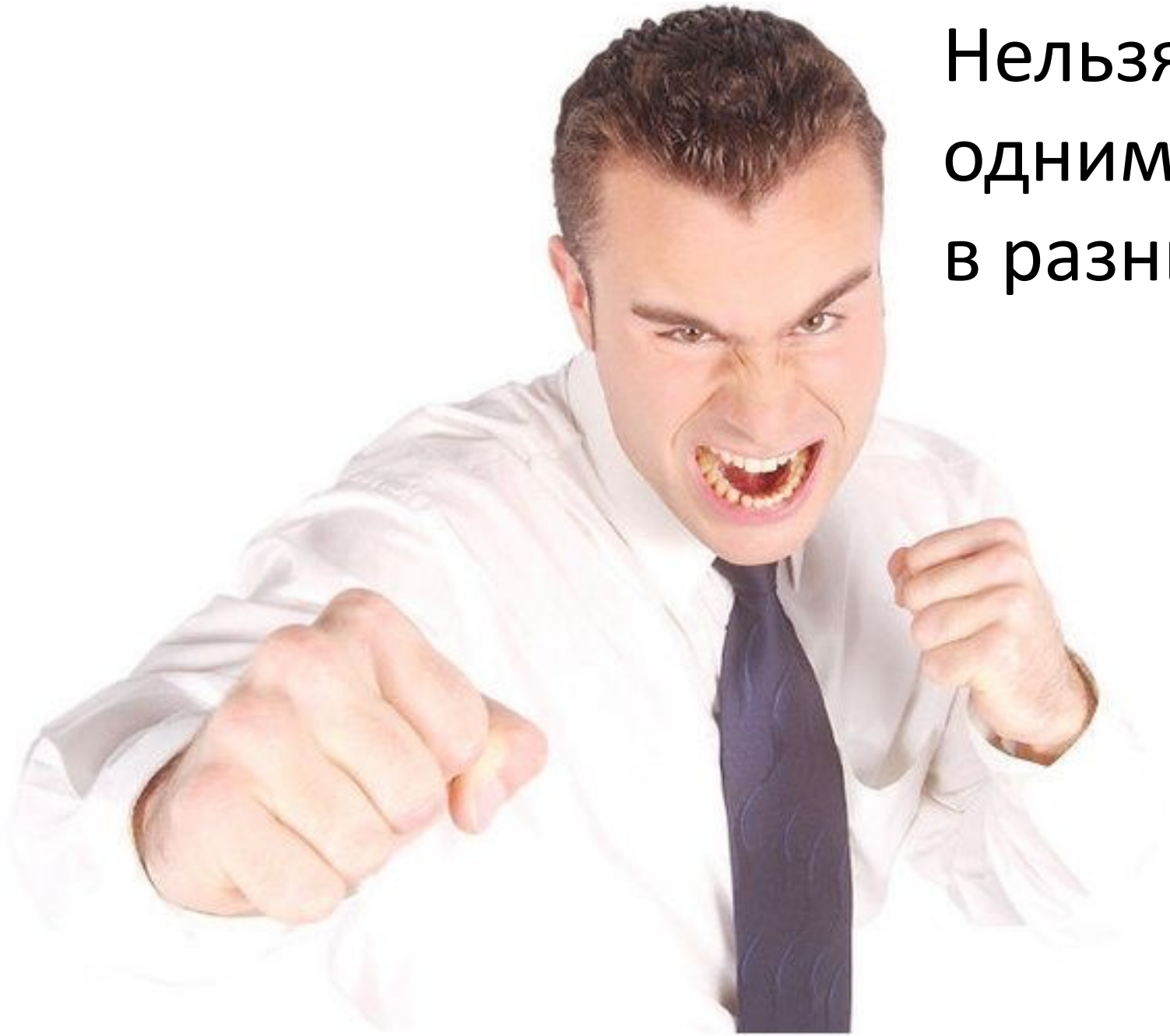
Ход размышлений:

1) Spring security – это цепочки фильтров и настройки конкретных фильтров.
2) Если мне надо, чтобы в одном случае сессия была, а в другом не было и все это одновременно, то необходимо два одинаковых фильтра работающих по разному или две разно настроенных цепочки фильтров.
3) Цепочка фильтров связана с помощи HttpSecurity.
4) Надо два объекта HttpSecurity.
5) Http-security получаем результате конфигурации
6) Значит, надо два конфига.
7) Спринг ругается и требует приоритет. Укажем приоритет.

```kotlin
@Order(1)
@Configuration
class SessionlessSecurityConfig(
    private val jpaUserDetailsServiceImpl: JPAUserDetailsServiceImpl,
    private val jwtTokenService: JWTTokenService
) : WebSecurityConfigurerAdapter(){

    override fun configure(http: HttpSecurity) {
        http
            .csrf().disable()
            .httpBasic().disable()
            .formLogin().disable()
            .sessionManagement().sessionCreationPolicy(SessionCreationPolicy.NEVER).and()
            .addFilterBefore(JWTTokenFilter(jwtTokenService), BasicAuthenticationFilter::class.java)
            .authorizeRequests()
            .antMatchers("/registration").permitAll()
            .antMatchers("/clients/all").hasRole(Role.MANAGER.name)
            .mvcMatchers("/clients/{id}").hasAnyRole(Role.MANAGER.name, Role.DOCTOR.name)
            .mvcMatchers("/visits/doctor/{id}").hasAnyRole(Role.MANAGER.name)
            .mvcMatchers("/employees/doctors").permitAll()
            .mvcMatchers("/auth/jwt/authenticate").permitAll()
            .mvcMatchers("/clients").MANAGER.name, Role.DOCTOR.name)
            .antMatchers("/clients").hasAnyRole(Role.hasAnyRole(Role.MANAGER.name, Role.DOCTOR.name)
            .anyRequest().authenticated()
    }
    // какие-то бины и настройки ниже
```

```kotlin
@Order(2)
@Configuration
class SessionFullSecurityConfig(
    private val jpaUserDetailsServiceImpl: JPAUserDetailsServiceImpl,
    private val passwordEncoder: PasswordEncoder
) : WebSecurityConfigurerAdapter() {

    override fun configure(http: HttpSecurity) {
        http.httpBasic().and()
            .formLogin().and()
            .sessionManagement().disable()
            .authorizeRequests { requests ->
                requests.anyRequest().authenticated()
            }
        http.authenticationProvider(usernamePasswordAuthenticationProvider())
    }

    override fun configure(auth: AuthenticationManagerBuilder) {
        auth.authenticationProvider(usernamePasswordAuthenticationProvider())
    }

    @Bean
    fun usernamePasswordAuthenticationProvider() =
        UsernamePasswordAuthenticationProvider(jpaUserDetailsServiceImpl,
passwordEncoder)
}
```


СЕССИЯ
ЗАКРЫТА!



Нельзя описать доступ к одним и тем же эндпоинтам в разных конфигах

Many applications have completely different access rules for one set of resources compared to another. For example, an application that hosts a UI and a backing API might support cookie-based authentication with a redirect to a login page for the UI parts and token-based authentication with a 401 response to unauthenticated requests for the API parts. Each set of resources has its own WebSecurityConfigurerAdapter with a unique order and its own request matcher. If the matching rules overlap, the earliest ordered filter chain wins.

FilterChainProxy

```
    Returns the first filter chain matching the supplied URL.
    Params: request – the request to match
    Returns: an ordered array of Filters defining the filter chain
private List<Filter> getFilters(HttpServletRequest request) {
    int count = 0;
    for (SecurityFilterChain chain : this.filterChains) {
        if (logger.isTraceEnabled()) {
            logger.trace(LogMessage.format( format: "Trying to match request against %s (%d/%d)", chain, ++count,
                    this.filterChains.size()));
        }
        if (chain.matches(request)) {
            return chain.getFilters();
        }
    }
    return null;
}
```

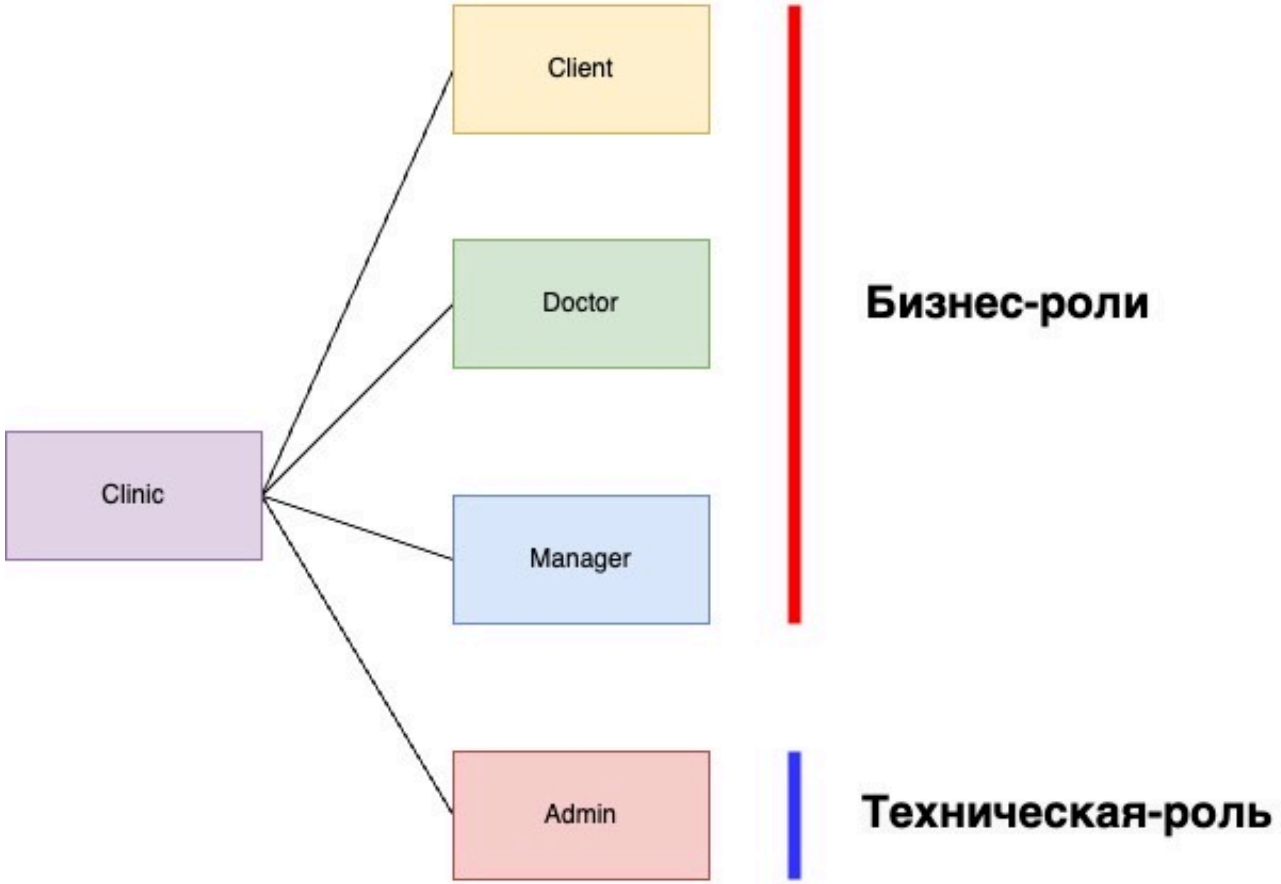
Разделение контроллеров:

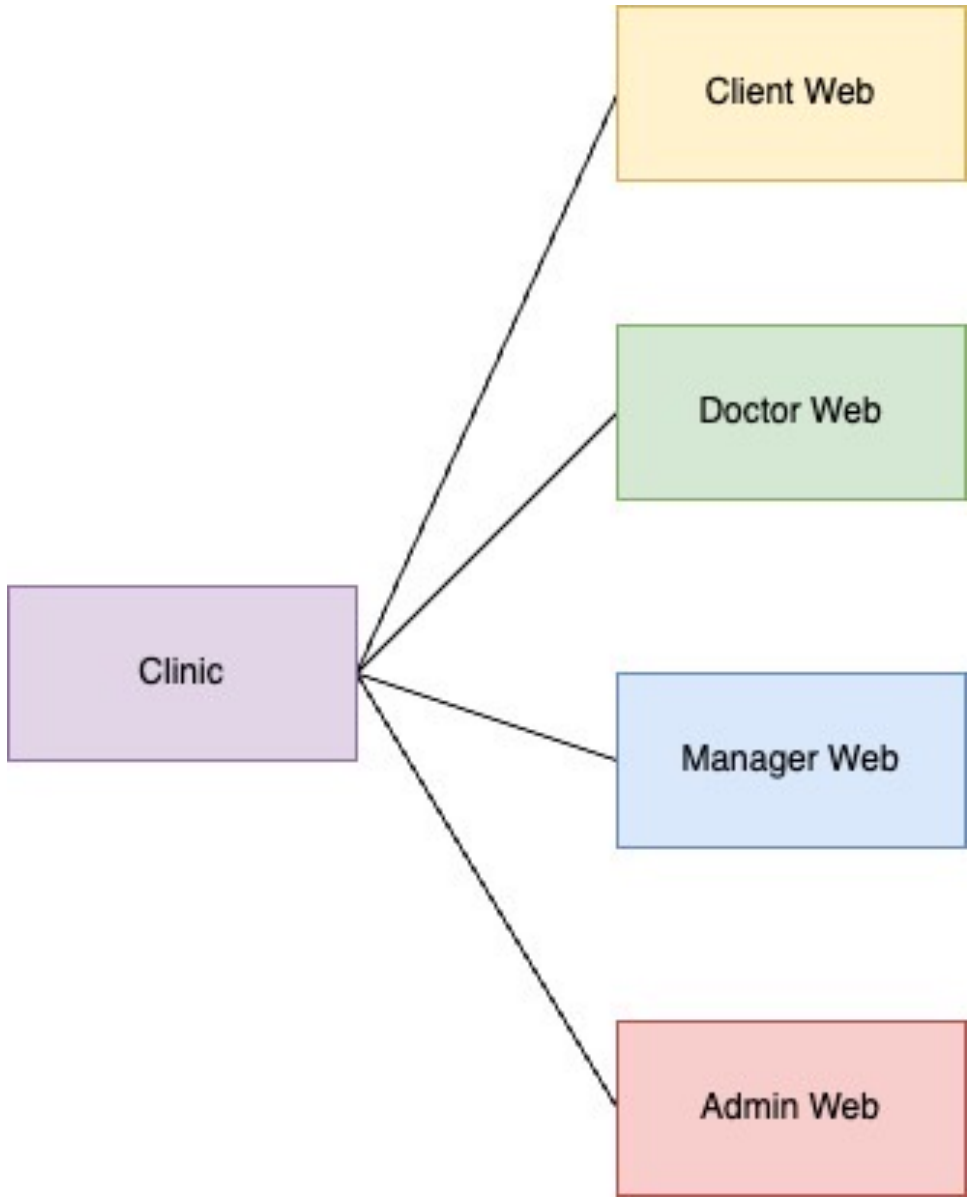
- /admin – для админа
- /doctor – для доктора
- /manager – для менеджера
- /client – для клиента

Разделение контроллеров:

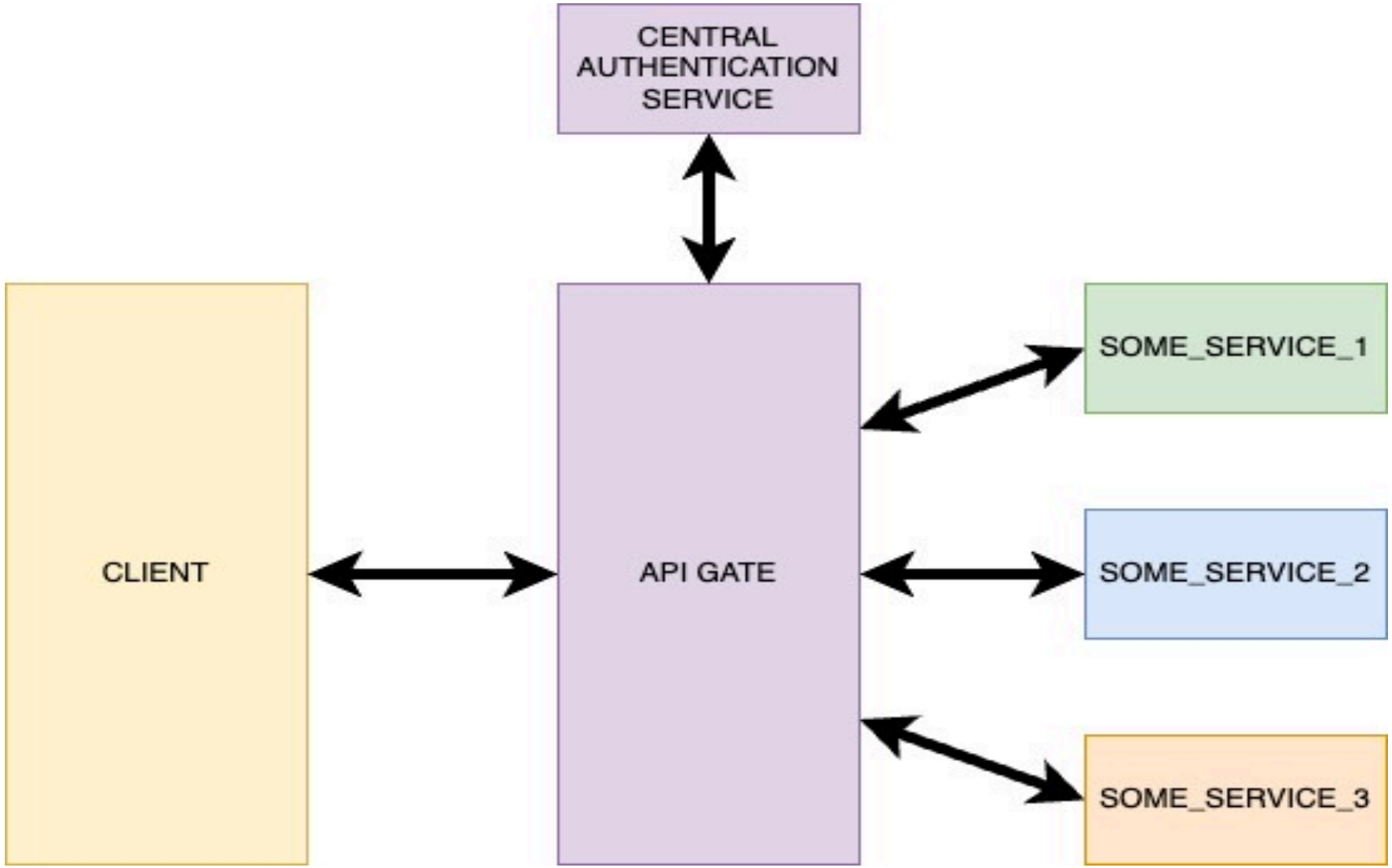
- /admin – для админа
- /doctor – для доктора
- /manager – для менеджера
- /client – для клиента

Разделение по ролям и продумывание вариантов использования

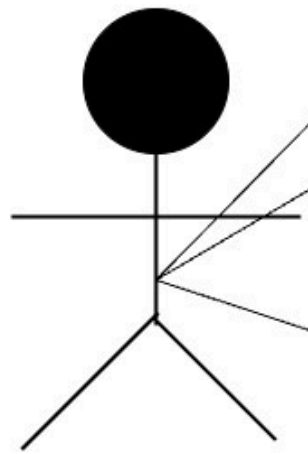



Client Web
Doctor Web
Clinic
Manager Web
Admin Web

Простейший пример CAS-архитектуры



1. CRM_OWNER
2. ROLE_CRM_2_HEAD
3.ROLE_CRM_HEAD
4. ROLE_CRM_2_MANAGER
5. ROLE_CRM_2_AUDITOR



Context 1

Client Web
CLIENT

Context 2

Another Manager Web
ROLE_CRM_2_MANAGER
ROLE_CRM_2_HEAD
ROLE_CRM_2_AUDITOR
ROLE_CRM_HEAD

CRM_2
CRM_2_USER
CRM_OWNER

Context 3

Manager Web
ROLE_CRM_MANAGER
ROLE_CRM_HEAD
ROLE_CRM_AUDITOR

CRM

ROLE_CRM_USER
CRM_OWNER

Резюме:

Пишем два конфига так, чтобы ресурсы в них не пересекались. Общие настройки выносим в configure-метод принимающий WebSecurity.

One more little important thing…

Самая …ITельная
IT компания

Роль

Смотреть

Роли и привилегии. Что хотел сказать автор Spring?

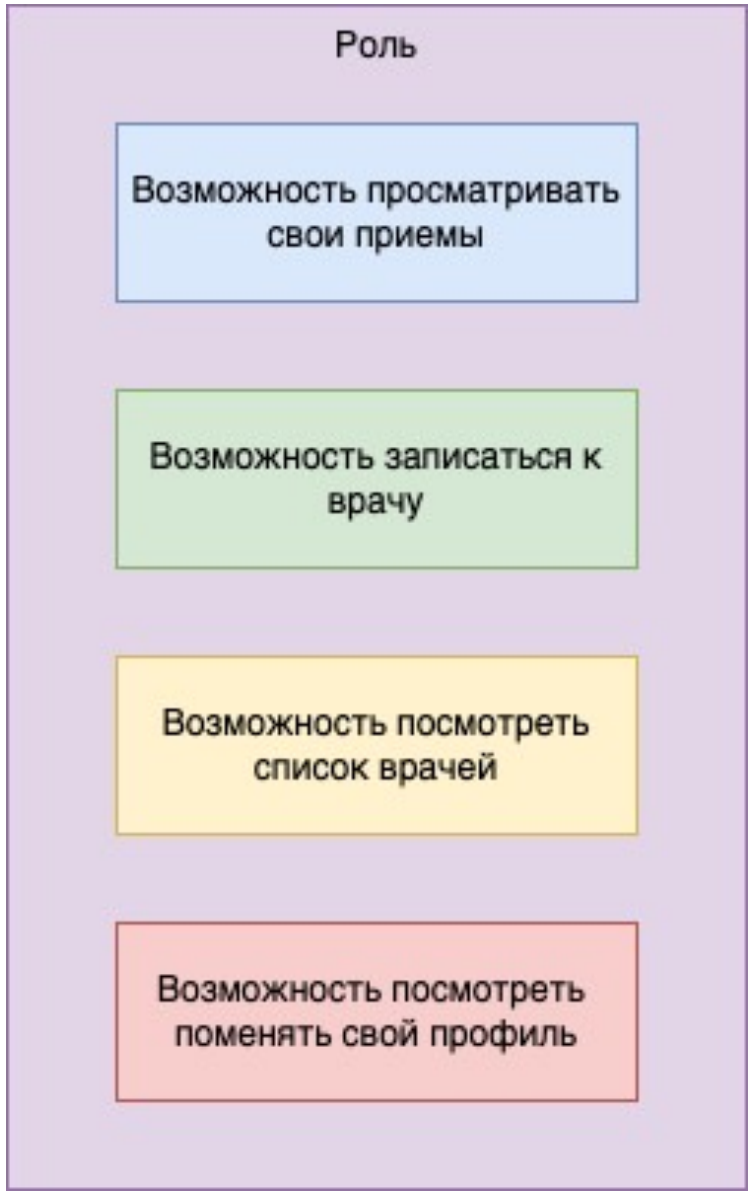
1. Единственное что есть в спринге для имплементации ролей – интерфейс GrantedAuthority и его имплементация SimpleGrantedAuthority.
2. С точки зрения Spring и роли и привилегии – это GrantedAuthority.
3. Роль по своей смысловой нагрузке – это набор привилегий.
4. Привилегия – это умение выполнять какое либо действие.
5. Интерфейс GrantedAuthority имеет один метод getAuthority() и он возвращает строку, потому что это универсальное представление, с которым удобно работать. Оно позволяет единообразно работать с ролями везде внутри SpringSecurity. Так работают RequestMatcher'ы в том числе.
6. Роли и привилегии желательно использовать раздельно. Либо роли, либо привилегии.
7. Вместе можно? – Можно, но с этим надо аккуратнее. Изначальное желание для каждой роли четко определить и прописать ее набор привилегий зачастую пагубно. Подумайте сто раз прежде чем использовать их вместе. Это красиво работает только в примерах. В жизни – это неудобно чаще всего.
8. Несколько ролей у пользователя – можно? Можно. Но нежелательно. Роль соответствует конкретному варианту использования. (Если ваш админ решил прийти в вашу челокилнику со своим человеком, то пусть создаст себе пользовательскую учетку.)
9. Как мы храним пользователей, роли, привилегии и т.д. в базе? Spring Security не диктует нам четких правил организации наших сущностей и связей между ними. Как хотим так и храним и связываем. Spring Security предлагает нам свою реализацию. На мой взгляд она неудобная.(Пример)
10. Роли – это строки или энамы? – как угодно, лишь бы getAuthority() правильно возвращал значение. Практика показывает, что при совместном использовании проще, если строки.

Роли и привилегии. Идеи после которых ваше приложение и его секьюрити конфиги превращаются в АД

1. Я хочу чтобы у меня была возможность сделать панель админа, где я мог бы для каждой роли выбрать набор привилегий из имеющихся и иметь возможность накидывать и снимать привилегии у ролей. Так можно? - Можно. Но с некоторыми танцами. В таком случае ваш конфиг стоит писать, основываясь скорее на привилегиях.
2. А можно чтобы при этом у пользователей были одновременно и роли и привилегии? – Можно. Это вопрос того, как организован ваш секьюрити конфиг и как организовано хранение ролей, привилегий и пользователей
3. Ты такой скептик…А зачем мне наличие одновременно и ролей и привилегий, если не для сценария из вопросов 1 и 2. – Такой вариант использования предполагает, что полномочия конкретных некоторых пользователей шире, чем просто роли.

В реальной жизни даже больших веб-проектов, таких, как Домклик, я не видел необходимости реализовывать функционал из пунктов 1 и 2. Обычно все сводится к наличию Одной роли, иногда нескольких ролей, очень редко к наличию роли и привилегии. Обычно, системы, которые управляют доступами и возможностями так тонко появляются очень редко. Использование наших B2C веб-приложений чаще всего позволяет обойтись без этого. Если необходимость реализовать такой функционал есть в вашем приложении – подумайте действительно ли он необходим.

Как нам кажется:

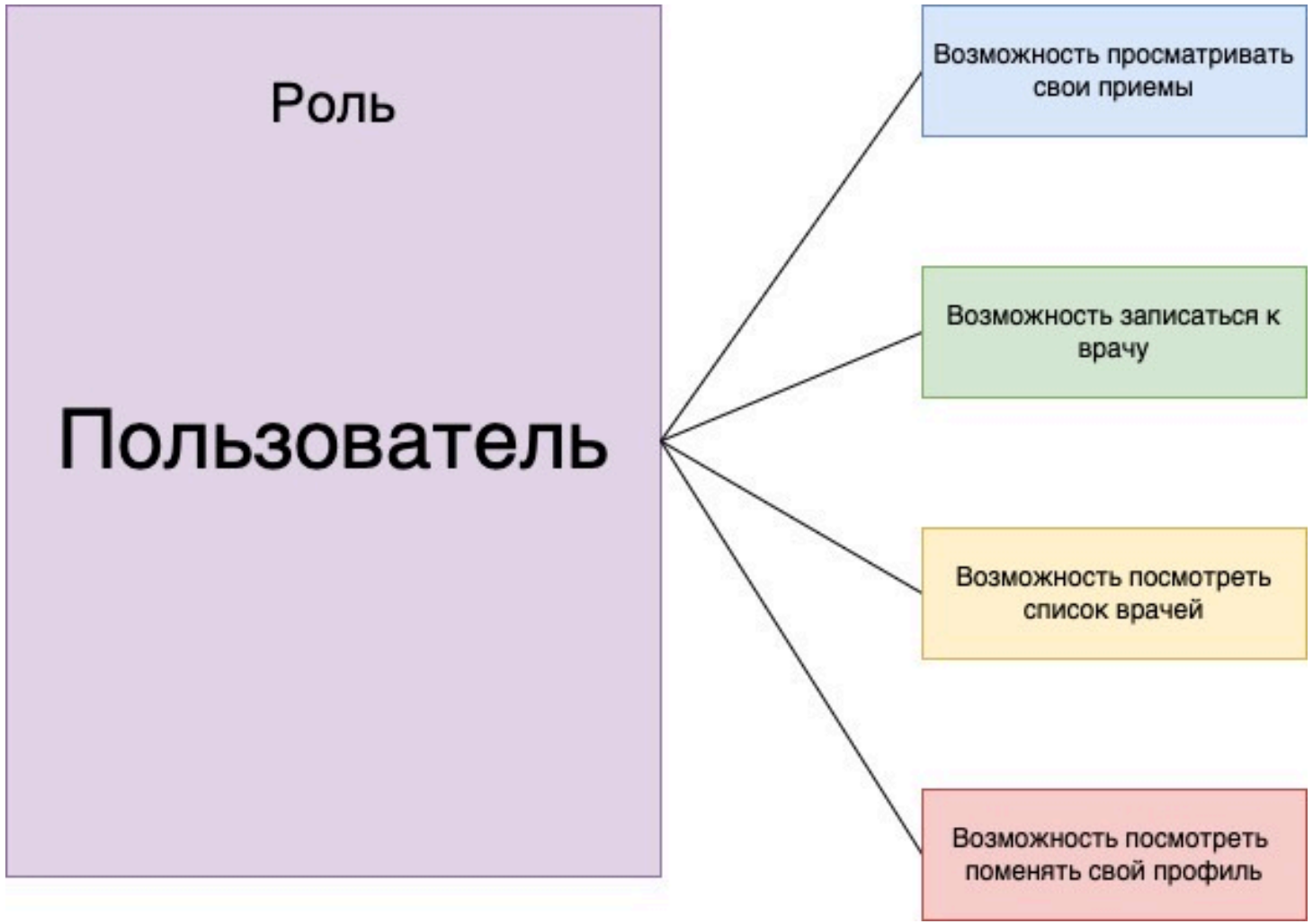


Как нам кажется:

Роль $\subset$ Возможность 1, Возможность 2…

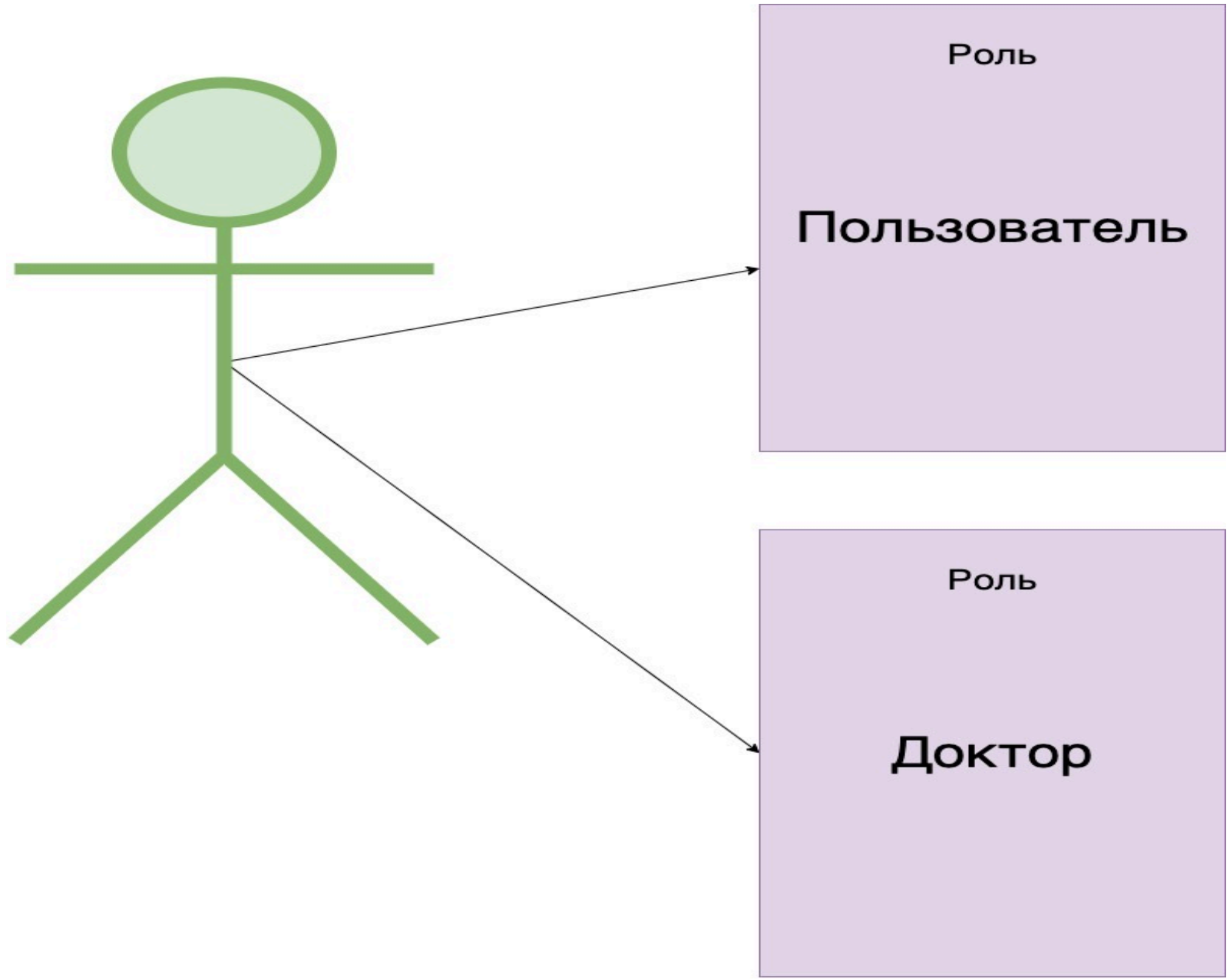
```kotlin
enum class Role(
    private val authority: String,
    private val permissions: List<Permission>
) : GrantedAuthority {
    MANAGER(
        "ROLE_MANAGER",
        listOf(Permission.SEE_ALL_CLIENTS, Permission.UPDATE_CLIENT, Permission.SEE_ALL_DOCTORS)
    );

    override fun getAuthority(): String {
        return this.authority
    }
}

enum class Permission : GrantedAuthority {
    SEE_ALL_CLIENTS, UPDATE_CLIENT, CREATE_CLIENT, DELETE_VISITS, SEE_ALL_DOCTORS, SEE_ALL_MANAGERS;

    override fun getAuthority(): String {
        return this.name
    }
}
```

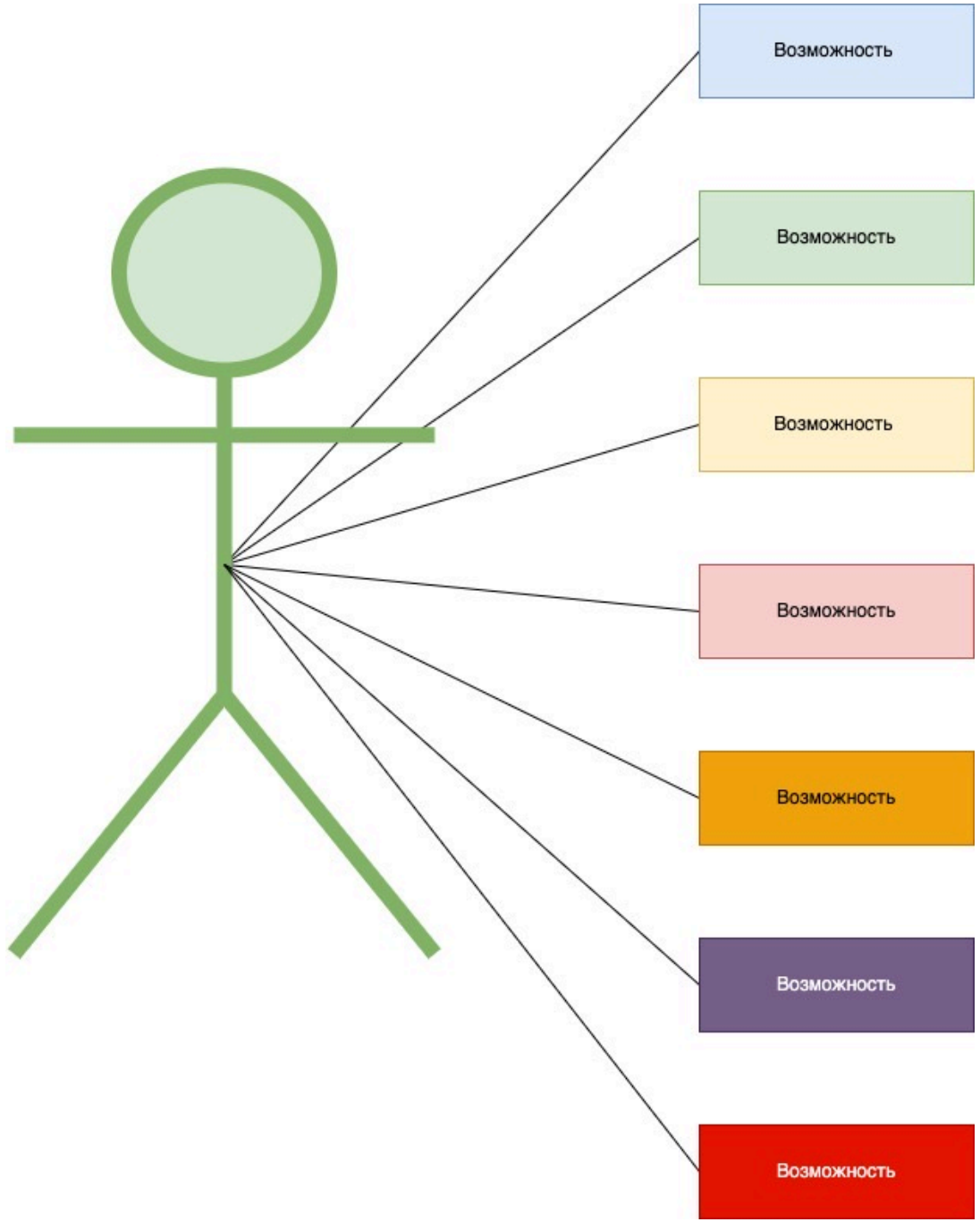
Как считает Spring:

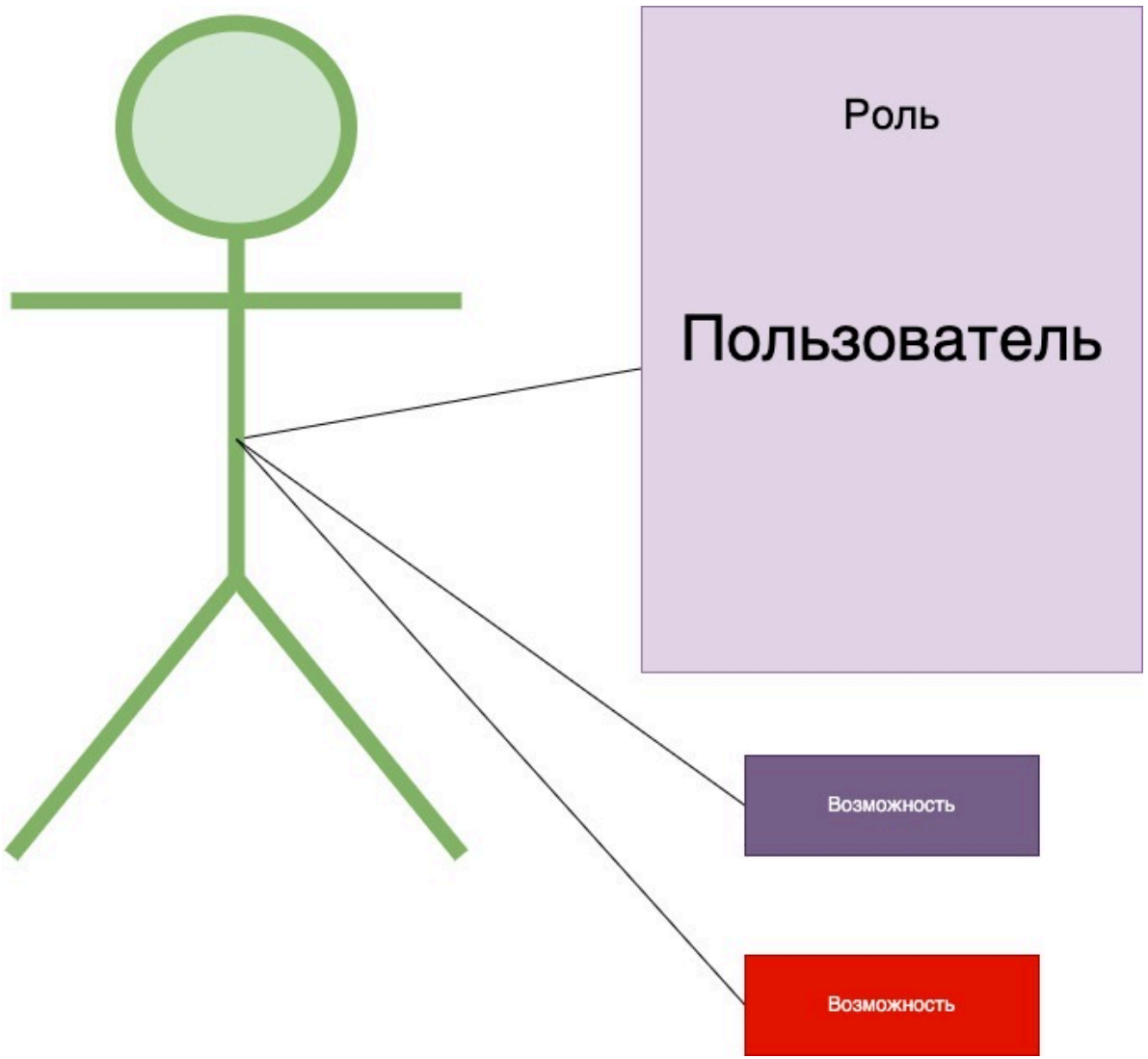


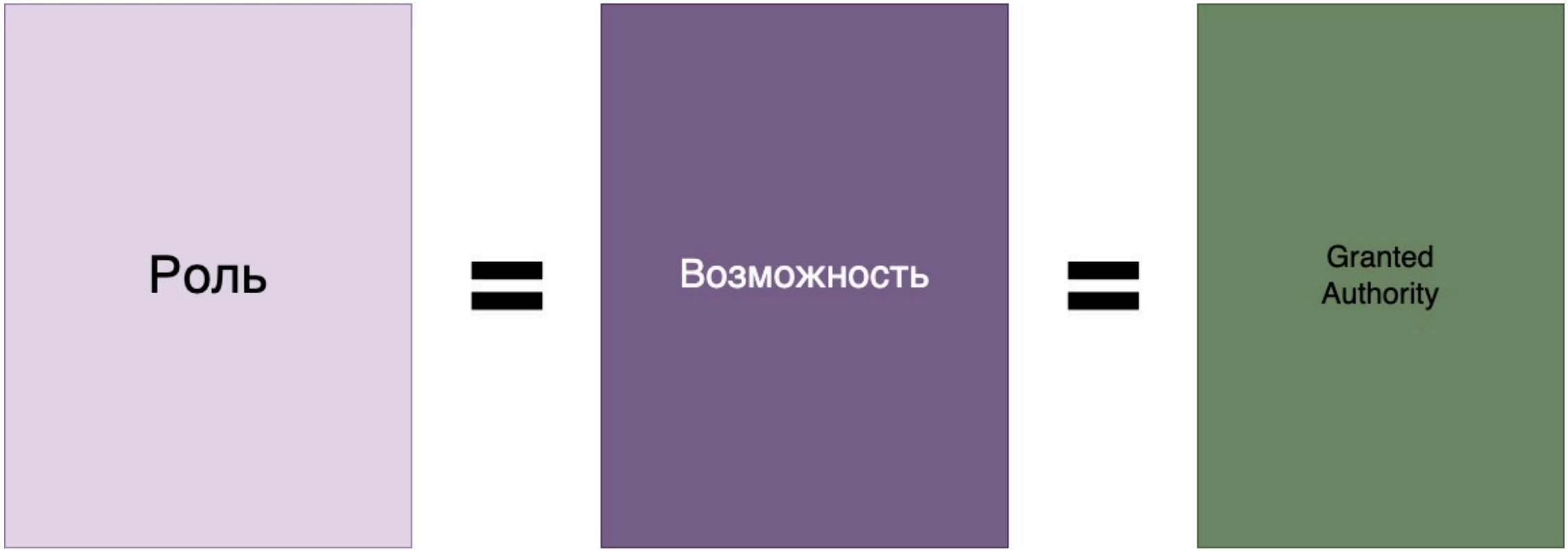
Как считает Spring:

Роль = Возможность 1 + Возможность 2 + Возможность 3


Роль
Пользователь
Роль
Доктор


Возможность
Возможность
Возможность
Возможность
Возможность
Возможность
Возможность


Роль
Пользователь
Возможность
Возможность


Роль
=
Возможность
=
Granted
Authority

hasRole	hasAurhority
USER	ROLE_USER
DOCTOR	ROLE_DOCTOR
MANAGER	ROLE_MANAGER

Role.getAuthority = "ROLE_" + ROLENAME

```
    Shortcut for specifying URLs require a particular role. If you do not want to have role prefix
    (default "ROLE_") automatically inserted see hasAuthority(String).

    Params: role - the role to require (i.e. USER, ADMIN, etc). Note, it should not start with role
            prefix as this is automatically inserted.

    Returns: the ExpressionUrlAuthorizationConfigurer for further customization
```

```
public ExpressionInterceptUrlRegistry hasRole(String role) {
    return access(ExpressionUrlAuthorizationConfigurer
            .hasRole(ExpressionUrlAuthorizationConfigurer.this.rolePrefix, role));
}
```

```
    Shortcut for specifying URLs require any of a number of roles. If you do not want to have role
    prefix (default "ROLE_") automatically inserted see hasAnyAuthority(String...)

    Params: roles - the roles to require (i.e. USER, ADMIN, etc). Note, it should not start with
            role prefix as this is automatically inserted.

    Returns: the ExpressionUrlAuthorizationConfigurer for further customization
```

```
public ExpressionInterceptUrlRegistry hasAnyRole(String... roles) {
    return access(ExpressionUrlAuthorizationConfigurer
            .hasAnyRole(ExpressionUrlAuthorizationConfigurer.this.rolePrefix, roles));
}
```

```
    Specify that URLs require a particular authority.

    Params: authority - the authority to require (i.e. ROLE_USER, ROLE_ADMIN, etc).

    Returns: the ExpressionUrlAuthorizationConfigurer for further customization
```

```
public ExpressionInterceptUrlRegistry hasAuthority(String authority) {
    return access(ExpressionUrlAuthorizationConfigurer.hasAuthority(authority));
}
```

Потерпи еще чуть!!! Будет про Spring Boot 3, новый Security, переезд на него, настройки, рекомендации и примеры кода.

Преисполняемся матчизма

```
.antMatchers("/registration").permitAll()
.antMatchers("/clients/all").hasRole(Role.MANAGER.name)
.mvcMatchers("/clients/{id}").hasAnyRole(Role.MANAGER.name, Role.DOCTOR.name)
.mvcMatchers("/visits/doctor/{id}").hasAnyRole(Role.MANAGER.name)
```

RequestMatchers

- **MvcMatchers** – принимает строковые ANT паттерны url
- **AntMatchers** - принимает строковые ANT паттерны url
- **RegexMatchers** – принимает строковые паттерны выражения

/clients

```
.antMatchers("/clients/all").hasRole(Role.MANAGER.name)
.mvcMatchers("/clients/all").hasRole(Role.MANAGER.name)
```

antMatchers - протектит только конкретный паттерн «/clients», а «/clients/» - нет.

mvcMatchers - защитит и «/clients» и «/clients/».

Важно протектить оба, так как для спрингового хендлера запросов они эквивалентны.

antMatcher(String antPattern) - Allows configuring the HttpSecurity to only be invoked when matching the provided ant pattern.

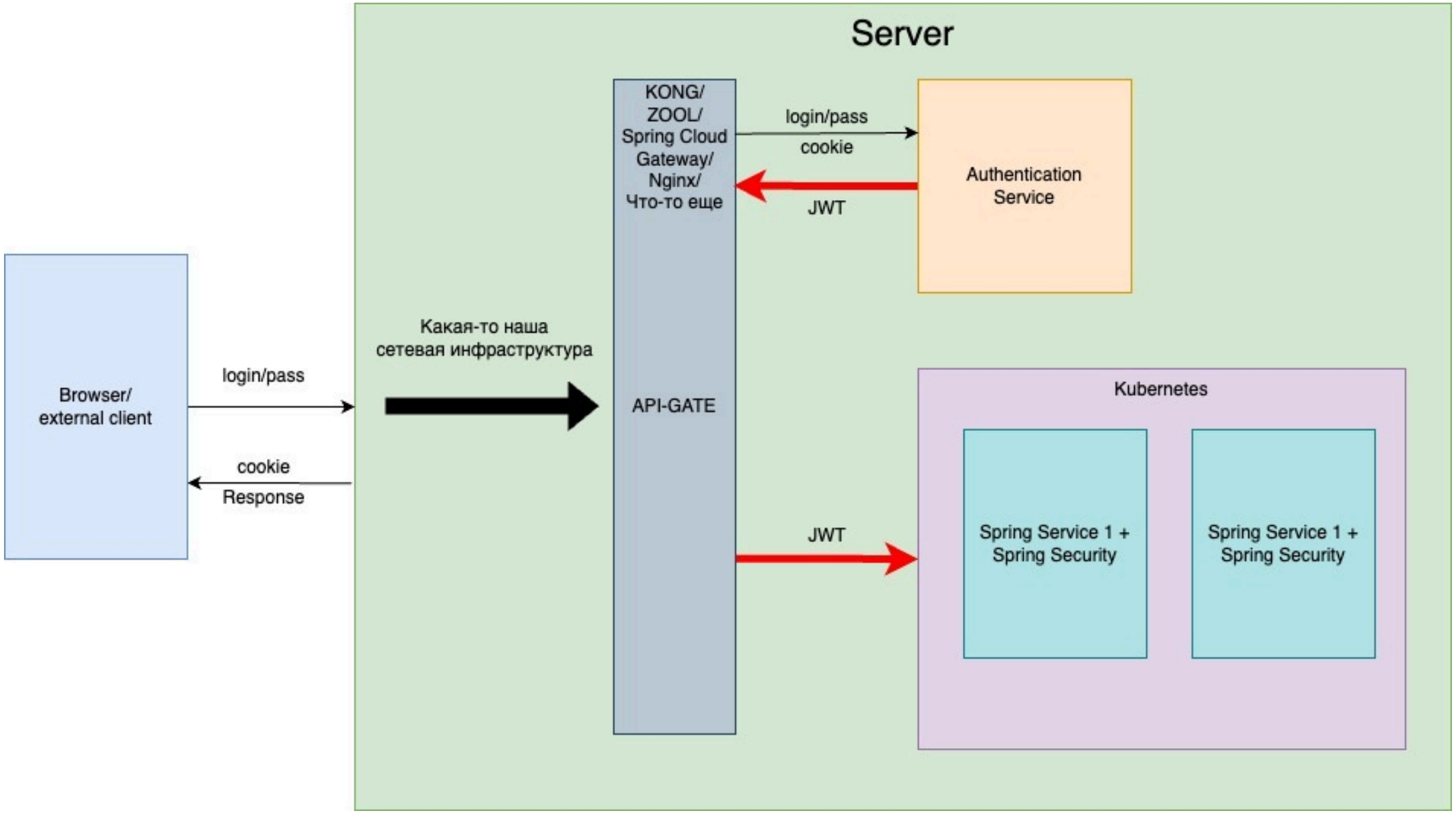
mvcMatcher(String mvcPattern) - Allows configuring the HttpSecurity to only be invoked when matching the provided Spring MVC pattern.

AntMatcher docs:

MvcRequestMatcher docs:

"/visits/doctor/{doctorId: regex}"

Jwt и nonblind токены при взаимодействии back to front. Можно ли?



Jwt и nonblind токены при взаимодействии back to front. Можно ли?

Михаил Вовренчук — OpenID Connect и OAuth2.0
Доклад про токены на фронте с Holy JS 2021.
Многое становится идейно яснее после просмотра.

Неплохая памятка про JWT в целом.

Статья на Хабре про то, как правильно хранить JWT на клиенте. Без мудреностей. Только База.

Где про новый Spring?


TYT

С какой версии все стало не таким, как раньше?

Начиная с версии 5.7, Spring Security стал меняться к текущему виду.

Текущая версия на момент доклада 6.1.4

На официальном сайте есть дока, которая готовит нас к 7.0, но совсем первичная и без дат релиза.

Пацаны из будущего

Spring Security
6.1.4

Spring Boot
3.0

Friday
Moscow

Spring под пиво для тех кому лениво

Обсуждаем Spring-новинки кратко и бегло.

Spring под пиво – лучший технический новостной блог для нетрудолюбивых, но технически настроенных аудиовизуалов.

Бегущая строка: Не разбирался сам и не разбираюсь. Ведь Павел Кислов из программы «Спринг под пиво» и так расскажет все самое важное.

LIVE
CNN
2:30 PM CET

JAVA 17 +

javax -> jakarta

~~WebSecurityConfigurerAdapter~~

```
@Configuration
class Configuration {
    @Bean
    fun securityFilterChain(http: HttpSecurity): SecurityFilterChain =
        http.httpBasic { }
            .formLogin { }
            .authorizeHttpRequests { requests ->
                requests.anyRequest().authenticated()
            }.build()

    @Bean
    fun webSecuritiyCustomizer(): WebSecurityCustomizer =
        WebSecurityCustomizer { web ->
            web.ignoring().requestMatchers("/ignore1", "/ignore2")
        }
}
```

Мелкие изменения.

- @Configuration убрали из аннотаций @EnableWebSecurity, @EnableMethodSecurity, @EnableGlobalMethodSecurity and @EnableGlobalAuthentication. **Теперь придется добавлять ее явно.**

- MVC, Regexp и Ant Matcher методы – заменены на универсальный RequestMatcher метод, который сам решает, что использовать, на основании того, использует ваше приложение Spring MVC или нет.

```
.requestMatchers("/visits/doctor/{doctorId}").hasAnyRole(Role.MANAGER.name)
.requestMatchers("/employees/doctors").authenticated()
.requestMatchers("/auth/jwt/authenticate").permitAll()
```

Мы подчиняемся документации и пишем, как нам предлагают. Запускаем иии падаем:

```
This method cannot decide whether these patterns are Spring MVC patterns or
not. If this endpoint is a Spring MVC endpoint, please use
requestMatchers(MvcRequestMatcher); otherwise, please use
requestMatchers(AntPathRequestMatcher).
```

```kotlin
@Bean
fun filterChainStateless(http: HttpSecurity, introspector: HandlerMappingIntrospector): SecurityFilterChain =
    introspector.let { it: HandlerMappingIntrospector
        val mvc = MvcRequestMatcher.Builder(it)
        http
            .sessionManagement { it: SessionManagementConfigurer<HttpSecurity!>!
                it.sessionCreationPolicy(SessionCreationPolicy.NEVER)
            }
            .formLogin { formLogin -> formLogin.disable() }
            .httpBasic { httpBasic -> httpBasic.disable() }
            .authenticationProvider(doctorUsernamePasswordAuthenticationProvider())
            .csrf { it.disable() }
            .addFilterBefore(JWTTokenFilter(jwtTokenService), BasicAuthenticationFilter::class.java)
            .authorizeHttpRequests { authorizeConfigurer ->
                authorizeConfigurer.requestMatchers(mvc.pattern("/registration/")).permitAll()
                    .requestMatchers(mvc.pattern("**/current")).permitAll()
                    .requestMatchers(mvc.pattern("/clients/all")).hasRole(Role.MANAGER.name)
                    .requestMatchers(mvc.pattern("/clients/{id}")).hasAnyRole(Role.MANAGER.name, Role.DOCTOR.name)
                    .requestMatchers(mvc.pattern("/visits/doctor/{doctorId}")).hasAnyRole(Role.MANAGER.name)
                    .requestMatchers(mvc.pattern("/employees/doctors")).authenticated()
                    .requestMatchers(mvc.pattern("/auth/jwt/authenticate")).permitAll()
                    .requestMatchers(mvc.pattern("/clients")).hasAnyRole(Role.MANAGER.name)
                    .requestMatchers(mvc.pattern("/h2-console/*")).hasRole(Role.ADMIN.name)
                    .anyRequest().authenticated()
            }.build() ^let
    }

@Bean
fun doctorUsernamePasswordAuthenticationProvider() =
        DoctorUsernamePaswordAuthenticationProvider(jpaUserDetailsSerivceImpl, passwordEncoder())

@Bean
fun passwordEncoder(): PasswordEncoder = NoOpPasswordEncoder.getInstance()
}
```

На эту тему в Spring зарепортили багу и она не пофикшена.

Вот так больше нельзя:

```
override fun configure(AuthenticationManagerBuilder auth) {
    auth.authenticationProvider(authenticationProvider)
}
```

Но все еще можно:

```
@Bean
fun authManager(httpSecurity: HttpSecurity): AuthenticationManager =
        httpSecurity
            .getSharedObject(AuthenticationManagerBuilder::class.java)
            .build()
```

Не стоит так делать. Использование один и тот же AuthenticationManager в разных FIlterChain без конкретной нужды - это пагубное явление, которое может привести к неочевидному поведению.

```kotlin
@Bean
@Throws(Exception::class)
fun authManager(http: HttpSecurity): AuthenticationManager? {
    val authenticationManagerBuilder = http.getSharedObject(AuthenticationManagerBuilder::class.java)
    authenticationManagerBuilder.authenticationProvider(doctorUsernamePasswordAuthenticationProvider())
    return authenticationManagerBuilder.build()
}
```

```kotlin
@Bean
fun filterChainStateless(http: HttpSecurity, introspector: HandlerMappingIntrospector): SecurityFilterChain =
    introspector.let { it: HandlerMappingIntrospector
        val mvc = MvcRequestMatcher.Builder(it)
        http
            .sessionManagement { it: SessionManagementConfigurer<HttpSecurity!>!
                it.sessionCreationPolicy(SessionCreationPolicy.NEVER)
            }
            .formLogin { formLogin -> formLogin.disable() }
            .httpBasic { httpBasic -> httpBasic.disable() }
            .authenticationProvider(doctorUsernamePasswordAuthenticationProvider())
```

Жизнь на Customizer

```
http.~~httpBasic~~().~~and~~() HttpSecurity!
    .~~formLogin~~().~~and~~()
    .~~sessionManagement~~().sessionCreationPolicy(SessionCreationPolicy.STATELESS)
    .~~and~~().~~csrf~~().disable() HttpSecurity!
```

```
http
    .sessionManagement { it: SessionManagementConfigurer<HttpSecurity!>!
        it.sessionCreationPolicy(SessionCreationPolicy.NEVER)
    }
    .formLogin { formLogin -> formLogin.disable() }
    .httpBasic { httpBasic -> httpBasic.disable() }
    .authenticationProvider(doctorUsernamePasswordAuthenticationProvider())
    .csrf { csrf -> csrf.disable() }
    .authorizeHttpRequests { authorizeConfigurer ->
        authorizeConfigurer.requestMatchers(mvc.pattern("/registration/")).permitAll()
            .requestMatchers(mvc.pattern("**/current")).permitAll()
            .requestMatchers(mvc.pattern("/clients/all")).hasRole(Role.MANAGER.name)
```

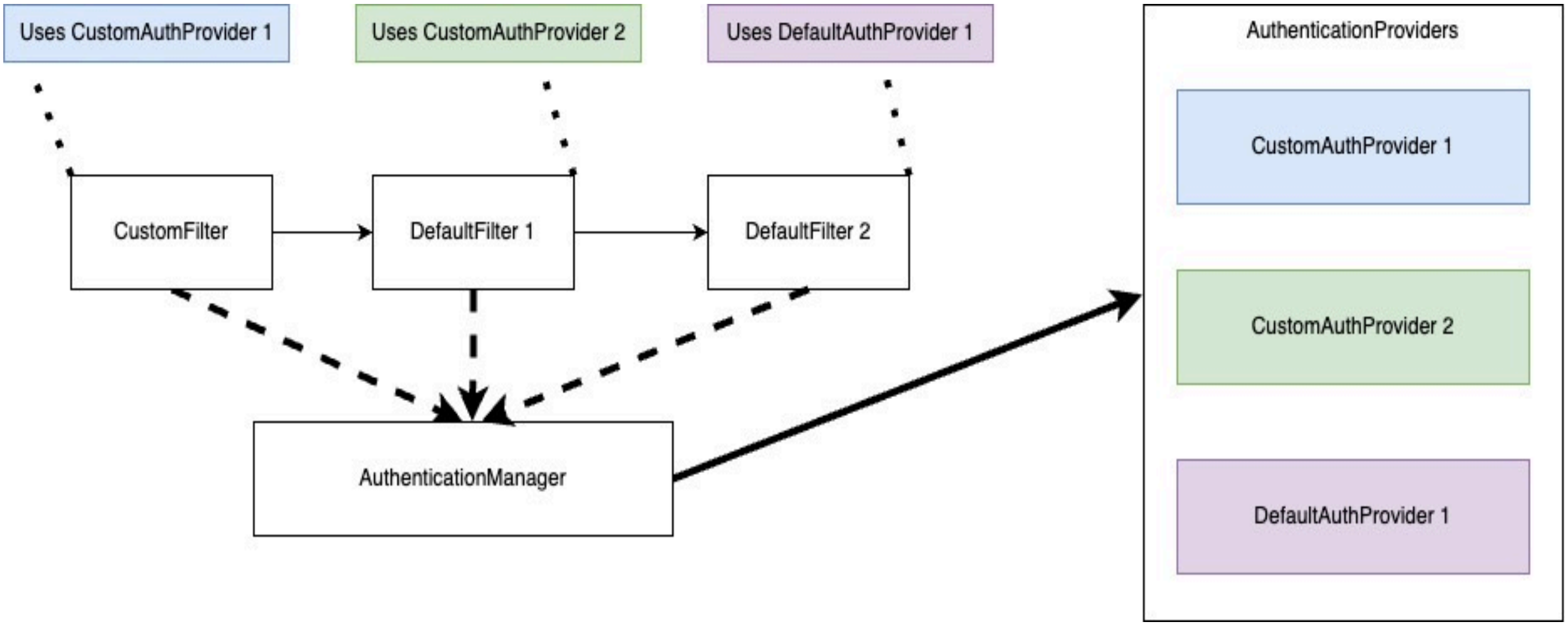
Не делаем так. Это только для примера.

```java
/**
 * Callback interface that accepts a single input argument and returns no result.
 *
 * @param <T> the type of the input to the operation
 * @author Eleftheria Stein
 * @since 5.2
 */
@FunctionalInterface
public interface Customizer<T> {

    /**
     * Performs the customizations on the input argument.
     * @param t the input argument
     */
    void customize(T t);

    /**
     * Returns a {@link Customizer} that does not alter the input argument.
     * @return a {@link Customizer} that does not alter the input argument.
     */
    static <T> Customizer<T> withDefaults() {
        return (t) -> {
        };
    }

}
```

```
http.httpBasic(Customizer.withDefaults())
    .formLogin{}
    .authorizeHttpRequests { requests ->
        requests.anyRequest().authenticated()
    }
```

Раньше:

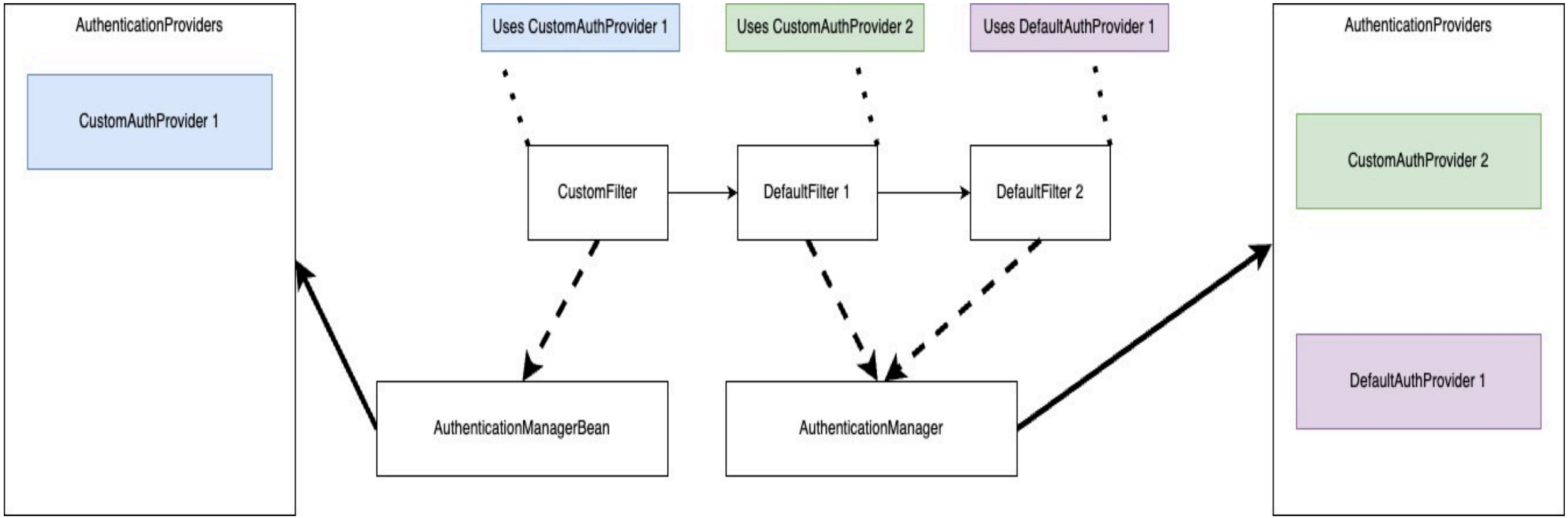
1) `configure(auth: AuthenticationManagerBuilder)`
2) `authenticationManagerBean(): AuthenticationManager`



Теперь:

~~1) configure(auth: AuthenticationManagerBuilder)~~
~~2) authenticationManagerBean(): AuthenticationManager~~

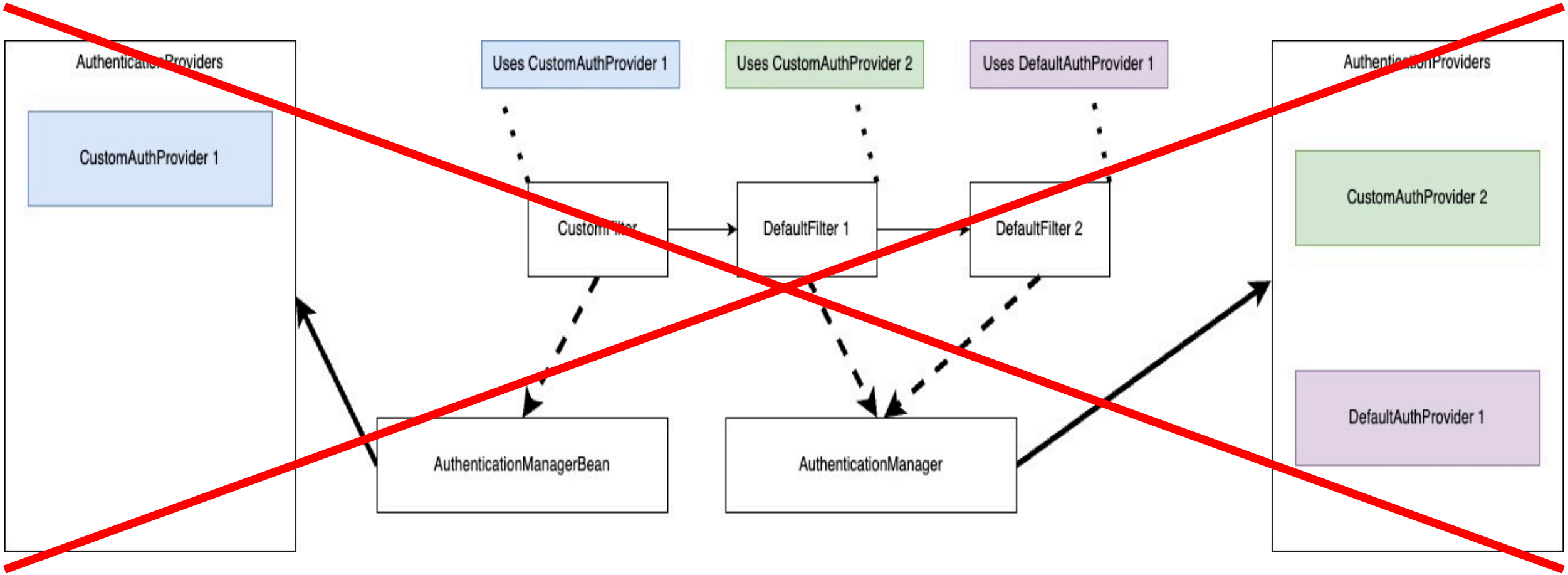
В интернете скажут:

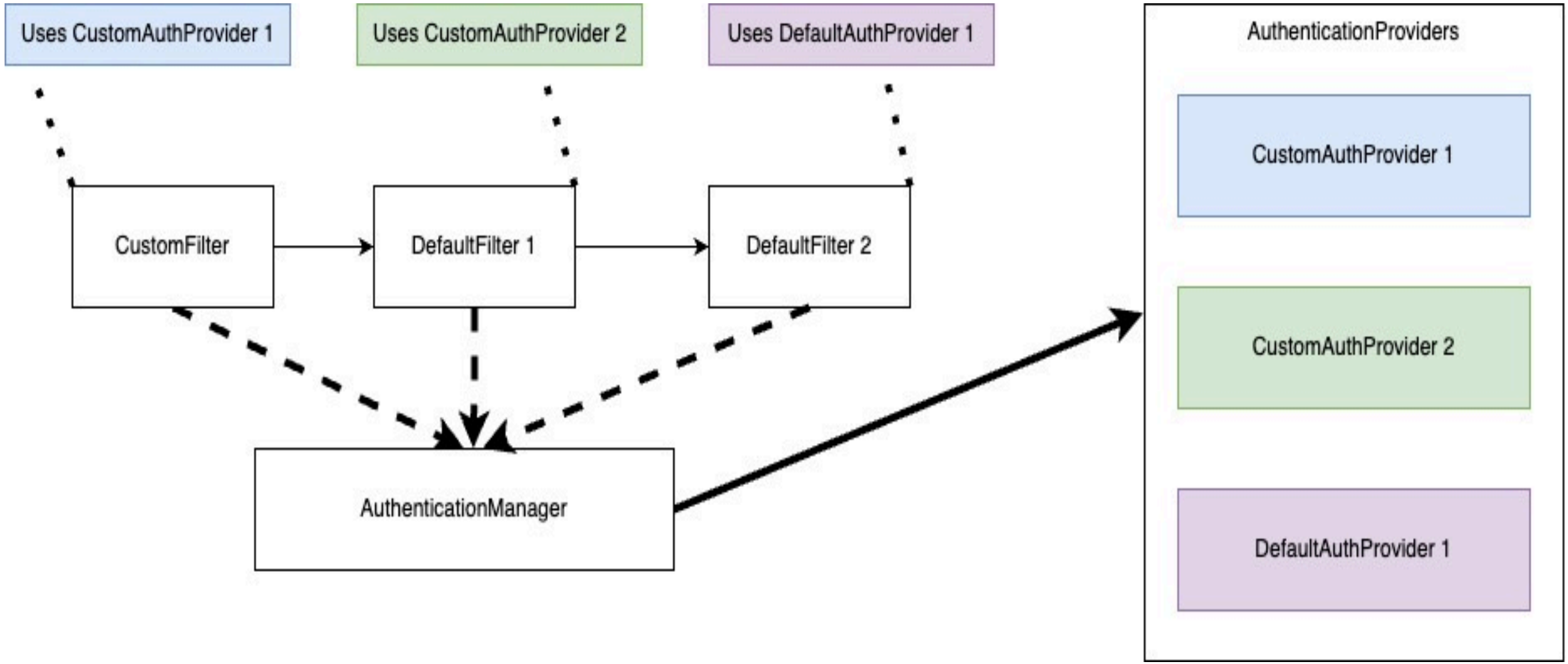


Теперь:

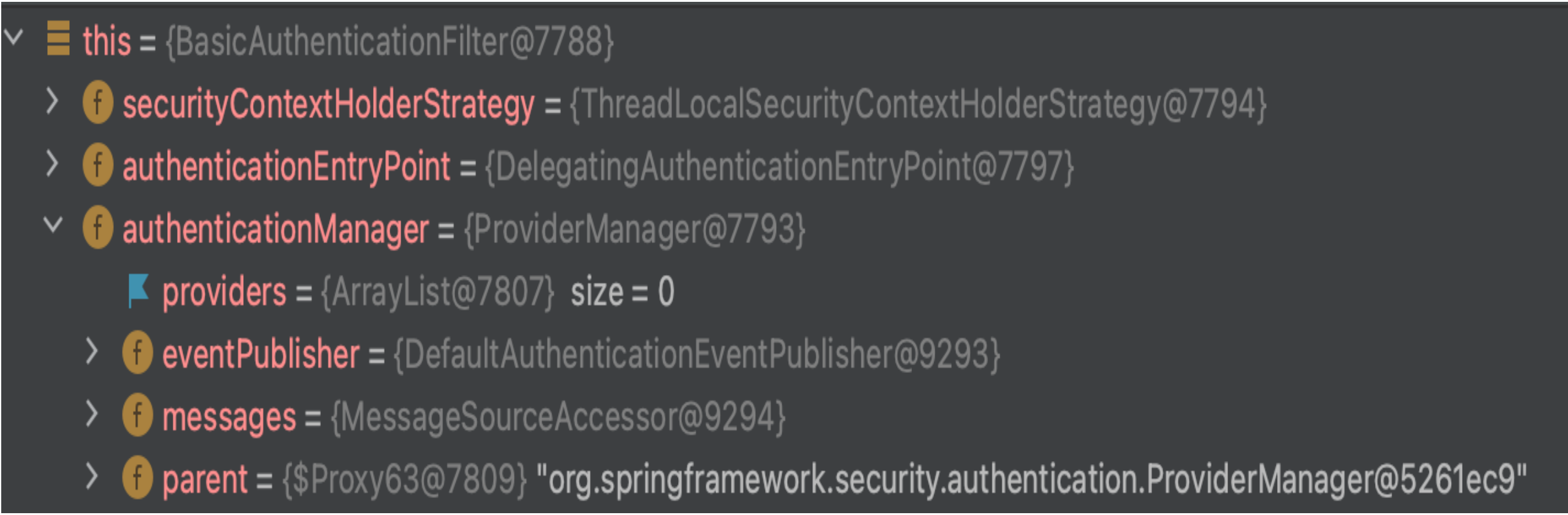
~~1) configure(auth: AuthenticationManagerBuilder)~~
~~2) authenticationManagerBean(): AuthenticationManager~~

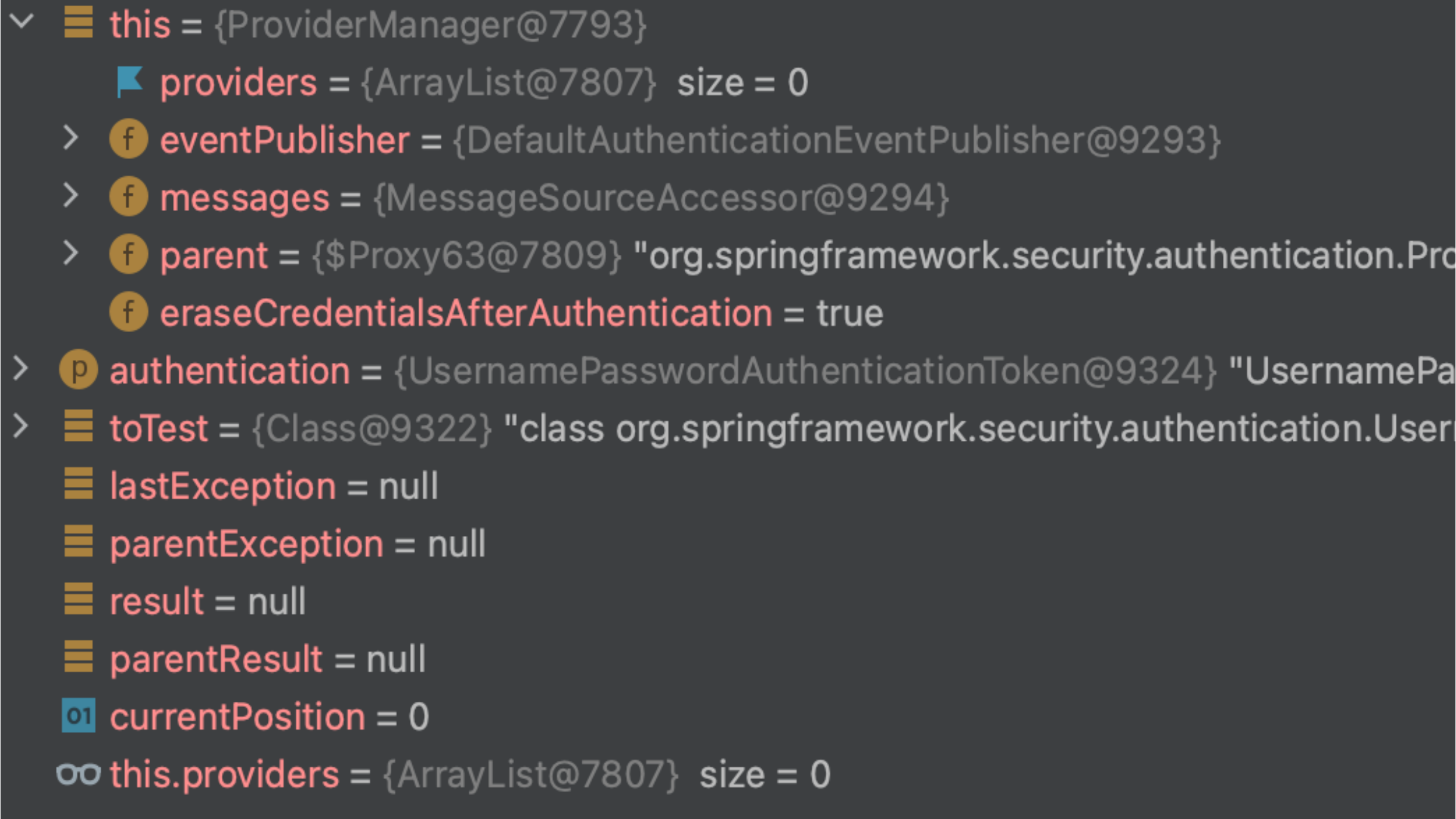
В интернете скажут:




Uses CustomAuthProvider 1
Uses CustomAuthProvider 2
Uses DefaultAuthProvider 1
CustomFilter
DefaultFilter 1
DefaultFilter 2
AuthenticationManager
AuthenticationProviders
CustomAuthProvider 1
CustomAuthProvider 2
DefaultAuthProvider 1

```kotlin
@Configuration
class Configuration {

    @Bean
    fun securityFilterChain (
            httpSecurity: HttpSecurity,
            authProvider1: UsernamePasswordAuthenticationProvider,
            authProvider2: SuperSecretHeaderAuthenticationProvider,
            authManager: AuthenticationManager
    ): SecurityFilterChain = httpSecurity
            .httpBasic {  }
            .authorizeHttpRequests{ it: AuthorizeHttpRequestsConfigurer<HttpSecurity!>.AuthorizationManagerRequestMatcherRegistry!
                it.anyRequest().authenticated()
            }
            .authenticationManager(authManager)
            .authenticationProvider(authProvider1)
            .authenticationProvider(authProvider2)
            .build()

    @Bean
    fun usernameAuthenticationProvider() = UsernamePasswordAuthenticationProvider()

    @Bean
    fun superSecretHeaderAuthenticationProvider() = SuperSecretHeaderAuthenticationProvider()

    @Bean
    fun authManager(httpSecurity: HttpSecurity): AuthenticationManager =
           httpSecurity.getSharedObject(AuthenticationManagerBuilder::class.java).build()



}
```


this = {BasicAuthenticationFilter@7788}
securityContextHolderStrategy = {ThreadLocalSecurityContextHolderStrategy@7794}
authenticationEntryPoint = {DelegatingAuthenticationEntryPoint@7797}
authenticationManager = {ProviderManager@7793}
providers = {ArrayList@7807} size = 0
eventPublisher = {DefaultAuthenticationEventPublisher@9293}
messages = {MessageSourceAccessor@9294}
parent = {$Proxy63@7809} "org.springframework.security.authentication.ProviderManager@5261ec9"


this = {ProviderManager@7793}
providers = {ArrayList@7807} size = 0
eventPublisher = {DefaultAuthenticationEventPublisher@9293}
messages = {MessageSourceAccessor@9294}
parent = {$Proxy63@7809} "org.springframework.security.authentication.Pro
eraseCredentialsAfterAuthentication = true
authentication = {UsernamePasswordAuthenticationToken@9324} "UsernamePa
toTest = {Class@9322} "class org.springframework.security.authentication.User
lastException = null
parentException = null
result = null
parentResult = null
currentPosition = 0
this.providers = {ArrayList@7807} size = 0

Debug: DemoApplication

Debugger Console Actuator

```
2023-09-27T11:04:26.949+03:00 TRACE 69859 --- [nio-8090-exec-1] w.c.HttpSessionSecurityContextRepository : Did not find SecurityContext in HttpSession BB92199A427F29AB22496B4F
2023-09-27T11:04:26.949+03:00 TRACE 69859 --- [nio-8090-exec-1] .s.s.w.c.SupplierDeferredSecurityContext : Created SecurityContextImpl [Null authentication]
2023-09-27T11:04:26.949+03:00 TRACE 69859 --- [nio-8090-exec-1] .s.s.w.c.SupplierDeferredSecurityContext : Created SecurityContextImpl [Null authentication]
2023-09-27T11:04:27.083+03:00 TRACE 69859 --- [nio-8090-exec-1] o.s.s.w.header.writers.HstsHeaderWriter  : Not injecting HSTS header since it did not match request to [Is Secu
2023-09-27T11:04:27.083+03:00 TRACE 69859 --- [nio-8090-exec-1] o.s.b.w.s.f.OrderedRequestContextFilter  : Cleared thread-bound request context: org.apache.catalina.connector.
2023-09-27T11:04:27.086+03:00 DEBUG 69859 --- [nio-8090-exec-1] o.a.c.loader.WebappClassLoaderBase       :     findClass(jdk.internal.reflect.GeneratedMethodAccessor4)
2023-09-27T11:04:27.086+03:00 DEBUG 69859 --- [nio-8090-exec-1] o.a.c.loader.WebappClassLoaderBase       :      findClassInternal(jdk.internal.reflect.GeneratedMethodAccessor
2023-09-27T11:04:27.088+03:00 DEBUG 69859 --- [nio-8090-exec-1] o.a.c.loader.WebappClassLoaderBase       :     --> Returning ClassNotFoundException
2023-09-27T11:04:27.088+03:00 DEBUG 69859 --- [nio-8090-exec-1] o.a.c.loader.WebappClassLoaderBase       :     --> Passing on ClassNotFoundException
2023-09-27T11:04:27.084+03:00 ERROR 69859 --- [nio-8090-exec-1] o.a.c.c.C.[.[.[/].[dispatcherServlet]    : Servlet.service() for servlet [dispatcherServlet] in context with pa

java.lang.StackOverflowError Create breakpoint : null
    at org.springframework.security.authentication.ProviderManager.authenticate(ProviderManager.java:172) ~[spring-security-core-6.1.4.jar:6.1.4]
    at jdk.internal.reflect.GeneratedMethodAccessor4.invoke(Unknown Source) ~[na:na] <2 internal lines>
    at org.springframework.aop.support.AopUtils.invokeJoinpointUsingReflection(AopUtils.java:343) ~[spring-aop-6.0.12.jar:6.0.12]
    at org.springframework.aop.framework.JdkDynamicAopProxy.invoke(JdkDynamicAopProxy.java:237) ~[spring-aop-6.0.12.jar:6.0.12]
    at jdk.proxy2/jdk.proxy2.$Proxy63.authenticate(Unknown Source) ~[na:na]
    at org.springframework.security.authentication.ProviderManager.authenticate(ProviderManager.java:201) ~[spring-security-core-6.1.4.jar:6.1.4]
    at jdk.internal.reflect.GeneratedMethodAccessor4.invoke(Unknown Source) ~[na:na] <2 internal lines>
    at org.springframework.aop.support.AopUtils.invokeJoinpointUsingReflection(AopUtils.java:343) ~[spring-aop-6.0.12.jar:6.0.12]
    at org.springframework.aop.framework.JdkDynamicAopProxy.invoke(JdkDynamicAopProxy.java:237) ~[spring-aop-6.0.12.jar:6.0.12]
    at jdk.proxy2/jdk.proxy2.$Proxy63.authenticate(Unknown Source) ~[na:na]
    at org.springframework.security.authentication.ProviderManager.authenticate(ProviderManager.java:201) ~[spring-security-core-6.1.4.jar:6.1.4]
    at jdk.internal.reflect.GeneratedMethodAccessor4.invoke(Unknown Source) ~[na:na] <2 internal lines>
    at org.springframework.aop.support.AopUtils.invokeJoinpointUsingReflection(AopUtils.java:343) ~[spring-aop-6.0.12.jar:6.0.12]
    at org.springframework.aop.framework.JdkDynamicAopProxy.invoke(JdkDynamicAopProxy.java:237) ~[spring-aop-6.0.12.jar:6.0.12]
    at jdk.proxy2/jdk.proxy2.$Proxy63.authenticate(Unknown Source) ~[na:na]
    at org.springframework.security.authentication.ProviderManager.authenticate(ProviderManager.java:201) ~[spring-security-core-6.1.4.jar:6.1.4]
    at jdk.internal.reflect.GeneratedMethodAccessor4.invoke(Unknown Source) ~[na:na] <2 internal lines>
    at org.springframework.aop.support.AopUtils.invokeJoinpointUsingReflection(AopUtils.java:343) ~[spring-aop-6.0.12.jar:6.0.12]
    at org.springframework.aop.framework.JdkDynamicAopProxy.invoke(JdkDynamicAopProxy.java:237) ~[spring-aop-6.0.12.jar:6.0.12]
    at jdk.proxy2/jdk.proxy2.$Proxy63.authenticate(Unknown Source) ~[na:na]
```

StackOverflowError

Version Control | Run | Debug | TODO | Database Changes | Problems | Spring | Terminal | Endpoints | Services | Profiler | Build | Dependencies

Breakpoint reached (a minute ago) 1183:1 LF UTF-8 4 spaces

А как это сделать?

```
@Configuration
class SecurityConfig(
    private val jpaUserDetailsSerivceImpl: JPAUserDetailsSerivceImpl
) {
    @Bean
    fun filterChain(http: HttpSecurity): SecurityFilterChain =
        http.httpBasic(Customizer.withDefaults())
            .formLogin(Customizer.withDefaults())
            .authorizeHttpRequests { requests ->
                requests.anyRequest().authenticated()
            }
            .authenticationManager(authManager(http)).build()

    fun usernamePasswordAuthenticationProvider() =
        UsernamePasswordAuthenticationProvider(jpaUserDetailsSerivceImpl, passwordEncoder())

    fun authManager(http: HttpSecurity): AuthenticationManager? {
        val authenticationManagerBuilder = http.getSharedObject(AuthenticationManagerBuilder::class.java)
        authenticationManagerBuilder.authenticationProvider(usernamePasswordAuthenticationProvider())
        return authenticationManagerBuilder.build()
    }

    fun passwordEncoder(): PasswordEncoder = NoOpPasswordEncoder.getInstance()
}
```

```kotlin
@Configuration
class SecurityConfig(
    private val jpaUserDetailsSerivceImpl: JPAUserDetailsSerivceImpl
) {
    @Bean
    fun filterChain(http: HttpSecurity): SecurityFilterChain =
        http.httpBasic(Customizer.withDefaults())
            .formLogin(Customizer.withDefaults())
            .authorizeHttpRequests { requests ->
                requests.anyRequest().authenticated()
            }.also { sec ->
                authManager(sec)
                    .also { manager ->
                        sec.authenticationManager(manager)
                        sec.addFilterBefore(
                            getMyCustomFilter(manager),
                            BasicAuthenticationFilter::class.java
                        )
                    }
            }
            .build()

    private fun getMyCustomFilter(authenticationManager: AuthenticationManager) = MyCustomFilter(authenticationManager)

    private fun usernamePasswordAuthenticationProvider() =
        UsernamePasswordAuthenticationProvider(jpaUserDetailsSerivceImpl, passwordEncoder())

    private fun authManager(http: HttpSecurity): AuthenticationManager {
        val authenticationManagerBuilder = http.getSharedObject(AuthenticationManagerBuilder::class.java)
        authenticationManagerBuilder.authenticationProvider(usernamePasswordAuthenticationProvider())
        return authenticationManagerBuilder.build()
    }

    private fun passwordEncoder(): PasswordEncoder = NoOpPasswordEncoder.getInstance()
}
```

1) Создали AuthenticationManager и заполнили его провайдерами

2) Добавили AuthenticationManager к HttpSecurity

3) Создали свой фильтр

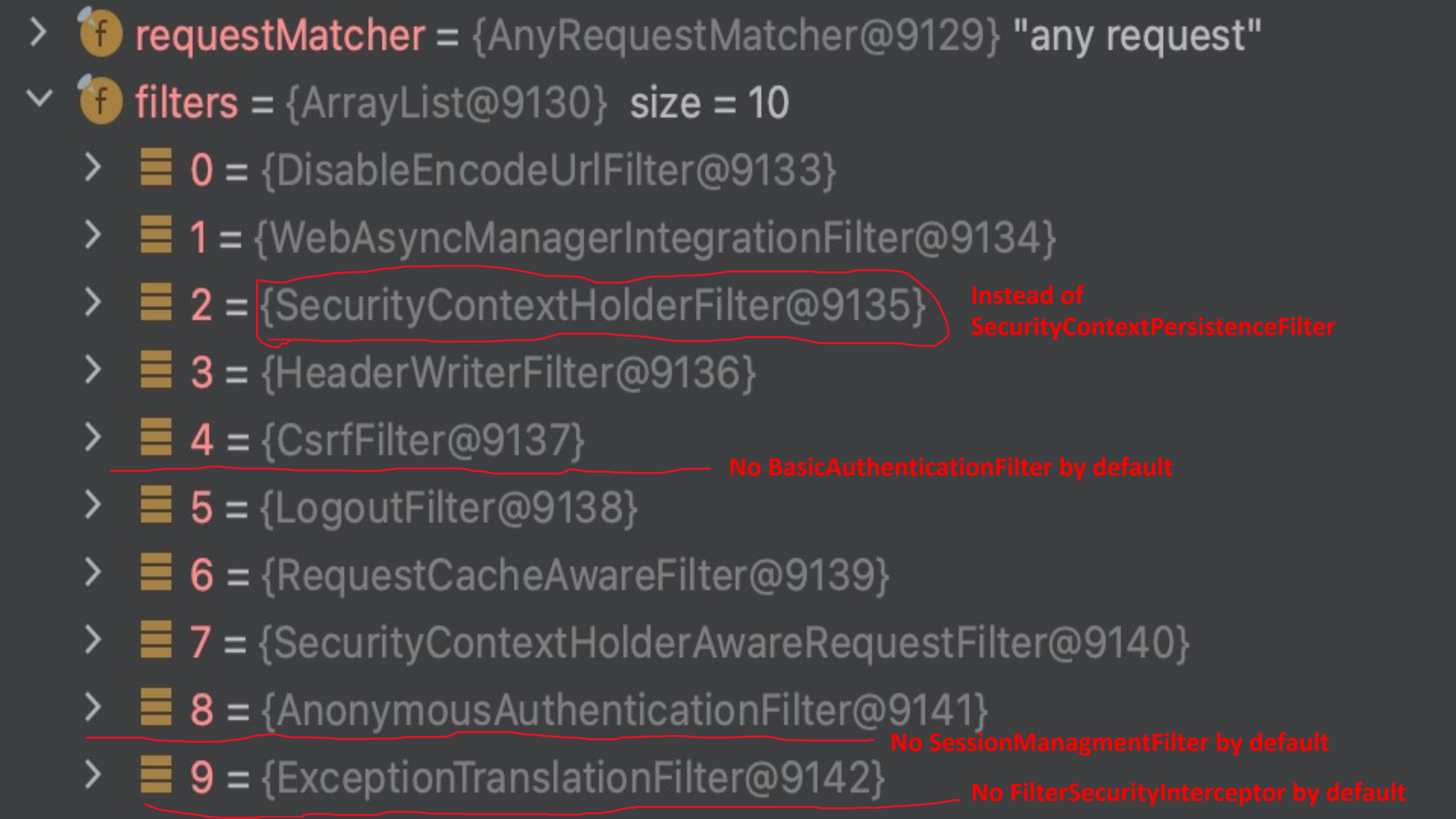
4) Добавили свой фильтр в цепочку

3

1

Spring Security

Уже не торт ☹


requestMatcher = {AnyRequestMatcher@9129} "any request"
filters = {ArrayList@9130} size = 10
0 = {DisableEncodeUrlFilter@9133}
1 = {WebAsyncManagerIntegrationFilter@9134}
2 = {SecurityContextHolderFilter@9135}
Instead of SecurityContextPersistenceFilter
3 = {HeaderWriterFilter@9136}
4 = {CsrfFilter@9137}
No BasicAuthenticationFilter by default
5 = {LogoutFilter@9138}
6 = {RequestCacheAwareFilter@9139}
7 = {SecurityContextHolderAwareRequestFilter@9140}
8 = {AnonymousAuthenticationFilter@9141}
No SessionManagmentFilter by default
9 = {ExceptionTranslationFilter@9142}
No FilterSecurityInterceptor by default

GET | localhost:8090/hello | Send

Params | Authorization | Headers (6) | Body | Pre-request Script | Tests | Settings | Cookies

Query Params

	Key	Value	Description	••• Bulk Edit
	Key	Value	Description	

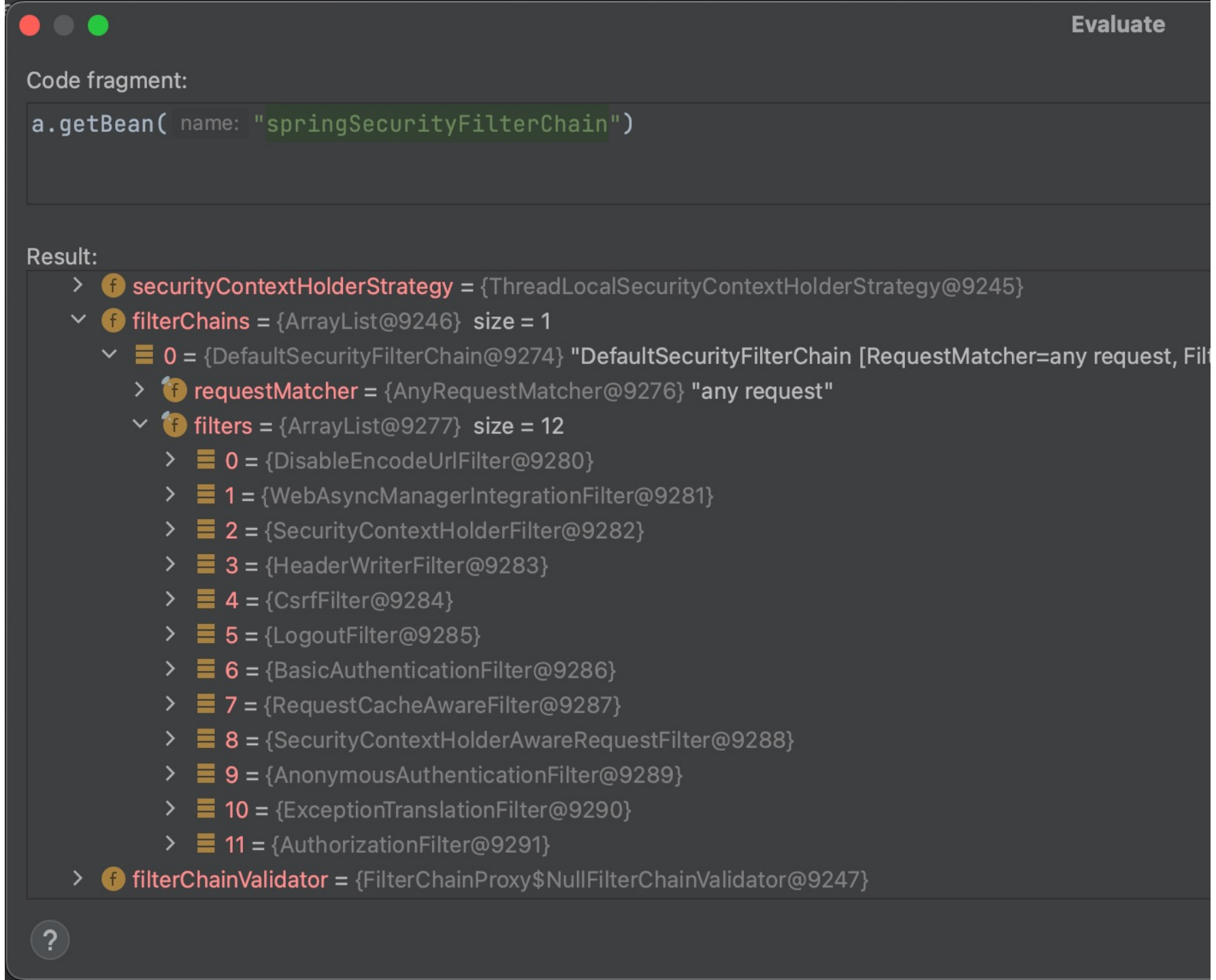
Body | Cookies | Headers (11) | Test Results | 200 OK | 87 ms | 338 B | Save as Example | •••

No cookies received from the server

All your cookies and their associated domains will appear here.

Postbot | Runner | Start Proxy | Cookies | Trash

```
@Bean
fun securityFilterChain(http: HttpSecurity): SecurityFilterChain =
    http.httpBasic {  }
        .formLogin{}
        .authorizeHttpRequests { requests ->
            requests.anyRequest().authenticated()
        }.build()
```


Evaluate
Code fragment:
a.getBean(name: "springSecurityFilterChain")
Result:
securityContextHolderStrategy = {ThreadLocalSecurityContextHolderStrategy@9245}
filterChains = {ArrayList@9246} size = 1
0 = {DefaultSecurityFilterChain@9274} "DefaultSecurityFilterChain [RequestMatcher=any request, Filt
requestMatcher = {AnyRequestMatcher@9276} "any request"
filters = {ArrayList@9277} size = 12
0 = {DisableEncodeUrlFilter@9280}
1 = {WebAsyncManagerIntegrationFilter@9281}
2 = {SecurityContextHolderFilter@9282}
3 = {HeaderWriterFilter@9283}
4 = {CsrfFilter@9284}
5 = {LogoutFilter@9285}
6 = {BasicAuthenticationFilter@9286}
7 = {RequestCacheAwareFilter@9287}
8 = {SecurityContextHolderAwareRequestFilter@9288}
9 = {AnonymousAuthenticationFilter@9289}
10 = {ExceptionTranslationFilter@9290}
11 = {AuthorizationFilter@9291}
filterChainValidator = {FilterChainProxy$NullFilterChainValidator@9247}
?

Что поменялось с фильтрами?

SecurityContext больше не сохраняется в репозиторий каждый раз по умолчанию, а только читается из него. Так как SecurityContextPersistenceFilter заменен на SecurityContextHolderFilter.

Если хотим вернуть старое поведение:

```
@Bean
fun securityFilterChain(http: HttpSecurity): SecurityFilterChain =
    http.httpBasic { }
        .formLogin { }
        .securityContext {
            it.requireExplicitSave(false)
        }
        .authorizeHttpRequests { requests ->
            requests.anyRequest().authenticated()
        }.build()
```

В таком случае будет использован привычный нам SecurityContextPersistenceFilter

Если нет:

Каждый раз, когда ваш фильтр сетит SecurityContext в SecurityContextHolder и вы хотите, чтобы он хранился в репозитории до следующего запроса, нужно инжектить в свой фильтр репозиторий и каждый раз явно вызывать

```
securityContextRepository.saveContext(context, request, response)
```

```java
@Override
protected void doFilterInternal(HttpServletRequest request, HttpServletResponse response, FilterChain chain)
        throws IOException, ServletException {
    try {
        UsernamePasswordAuthenticationToken authRequest = this.authenticationConverter.convert(request);
        if (authRequest == null) {
            this.logger.trace("Did not process authentication request since failed to find "
                    + "username and password in Basic Authorization header");
            chain.doFilter(request, response);
            return;
        }
        String username = authRequest.getName();
        this.logger.trace(LogMessage.format("Found username '%s' in Basic Authorization header", username));
        if (authenticationIsRequired(username)) {
            Authentication authResult = this.authenticationManager.authenticate(authRequest);
            SecurityContext context = this.securityContextHolderStrategy.createEmptyContext();
            context.setAuthentication(authResult);
            this.securityContextHolderStrategy.setContext(context);
            if (this.logger.isDebugEnabled()) {
                this.logger.debug(LogMessage.format("Set SecurityContextHolder to %s", authResult));
            }
            this.rememberMeServices.loginSuccess(request, response, authResult);
            this.securityContextRepository.saveContext(context, request, response);
            onSuccessfulAuthentication(request, response, authResult);
        }
    }
    catch (AuthenticationException ex) {
        this.securityContextHolderStrategy.clearContext();
        this.logger.debug( message: "Failed to process authentication request", ex);
        this.rememberMeServices.loginFail(request, response);
        onUnsuccessfulAuthentication(request, response, ex);
        if (this.ignoreFailure) {
            chain.doFilter(request, response);
        }
        else {
            this.authenticationEntryPoint.commence(request, response, ex);
        }
```

BasicAuthenticationFilter и его новый код, подтверждающий вышесказанное

ну и добивочка.

Ссылка на официальный
гайд по миграции на 6.0

Spring Security, demystified
by Daniel Garnier Moiroux

Everything new in Spring Security 6
baked with a Spring Boot 3
recipe by Laur Spilca @ Spring I/O

Spring Security Fundamentals 2022

Spring Security in Action

Спасибо!!!

Реальная благодарочка!!!

Спасибо!!!


тогда
я устал, я ухожу