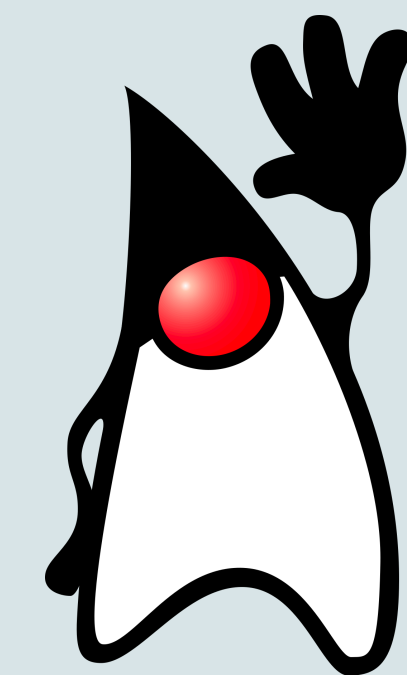




СБЕРБАНК

Корпоративный
университет



Spring Security

Занятие №17



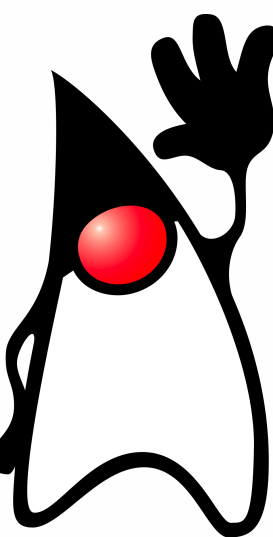
СБЕРБАНК

Корпоративный
университет

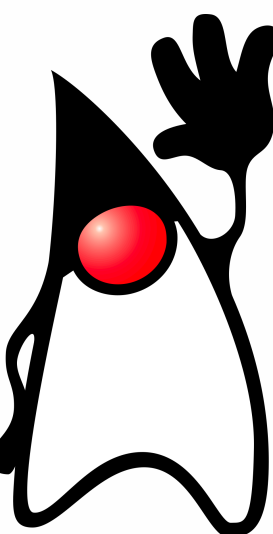


Что сможем после занятия

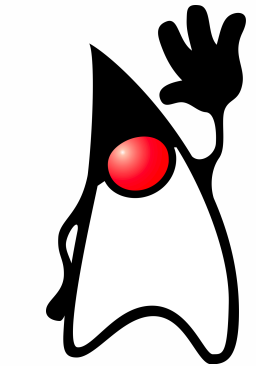
- Задачи безопасности
- Архитектура Spring Security
- Spring Security и Web
- Аутентификация
- Авторизация по URL



- Активно участвуем. Не стесняйтесь задавать вопрос.
- Но off-topic обсуждаем в Telegram @sb_ku_java_2019_10
- Не стесняйтесь просто спрашивать в telegram.
-

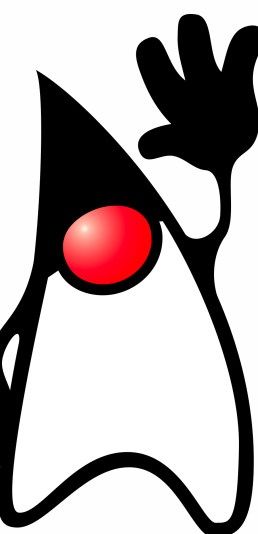


**Договорились?
Поехали!**



- **Задачи безопасности**

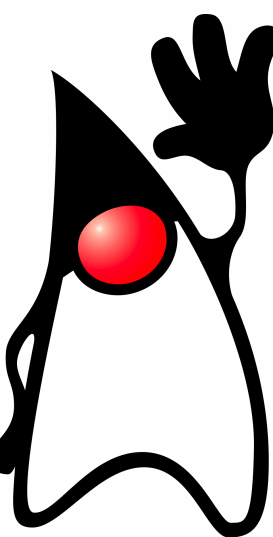
- Архитектура Spring Security
- Spring Security и Web
- Аутентификация
- Авторизация по URL



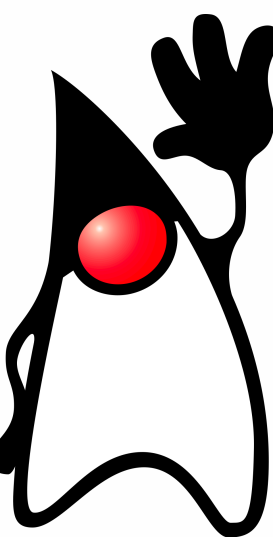
00

**Безопасность
приложений**

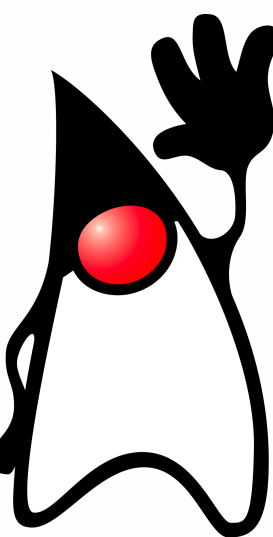
- Пользователь тот, за кого он себя выдает
- Пользователю предоставляет доступ только к тому функционалу, к которому он имеет доступ



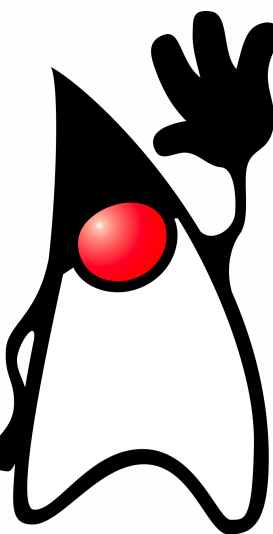
- Механизм подтверждения подлинности пользователя
- Механизм предоставления/запрещения доступа пользователям
- Механизм хранения прав доступа
- Механизм проверки прав доступа



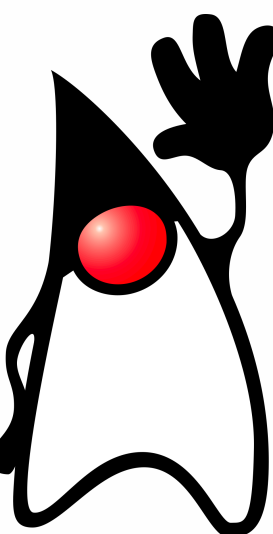
- Аутентификация - проверка что пользователь тот, за кого себя выдает
- Авторизация - проверка/предоставление прав доступа к объекту (объекты бывают разные)



- По URL-ам
- Методы в сервисах
- По объектам

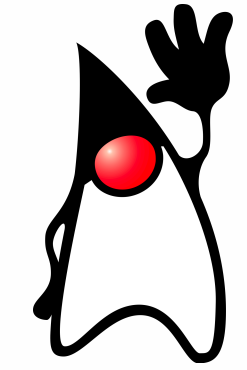


- Не является Firewall
- Не является антивирусом
- Не спасает приложение от различных атак - SQL Injection, XSS и т. п.
- Хотя есть защита от некоторых
- Не шифрует/обфусцирует поток



Ваши вопросы?

- Задачи безопасности

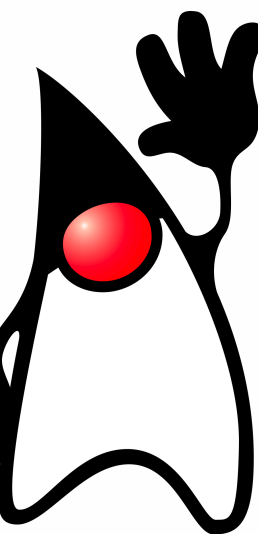


- **Архитектура Spring Security**

- Spring Security и Web

- Аутентификация

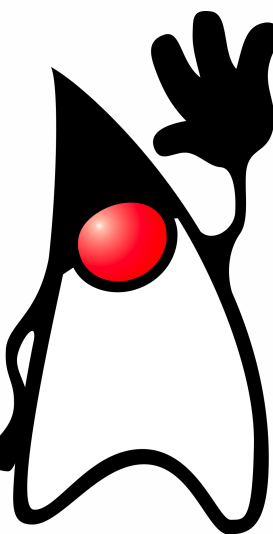
- Авторизация по URL



01

Архитектура Spring Security

- **spring-security-core** – содержит основные абстракции
- **spring-security-web** – содержит дополнительные классы, для работы в Web-окружении (Servlets)
- **spring-security-config** – для описания конфигурации Spring Security с помощью Spring Security XML
- **spring-security-web + spring-security-config** = минимальный набор зависимостей
- **spring-security-acl** – содержит описание абстракций ACL (авторизация)
- **spring-security-ldap** – LDAP
- **spring-security-openid** – интеграция по протоколу OAuth 2.0 + OpenID Connect
- **spring-security-test** – классы для unit-тестирования



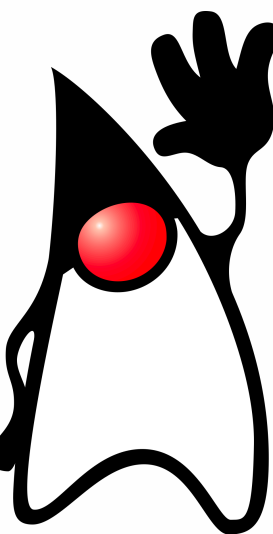
SecurityContext

Authentication

UserDetails

UserDetailsService

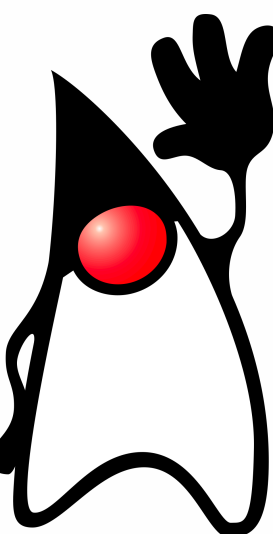
GrantedAuthority



Интерфейс SecurityContext

Отражает текущий контекст безопасности

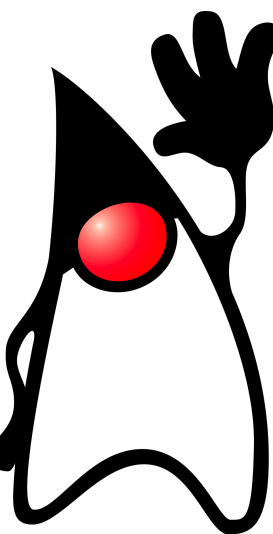
Является контейнером для объекта типа Authentication



MODE_THREADLOCAL

MODE_INHERITABLETHREADLOCAL

MODE_GLOBAL



Authentication – отражает информацию о текущем пользователе и его привилегиях

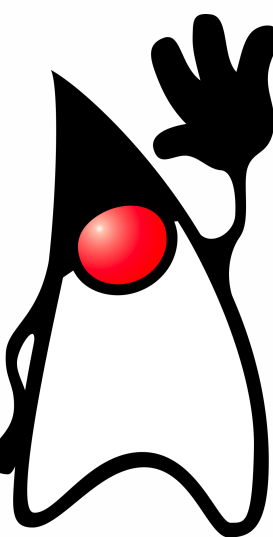
Principal:

В терминал логин-пароль – это логин

Отражает учетную запись пользователя

В Authentication представлено объектом типа Object

Зачастую реализуется объектом типа UserDetails

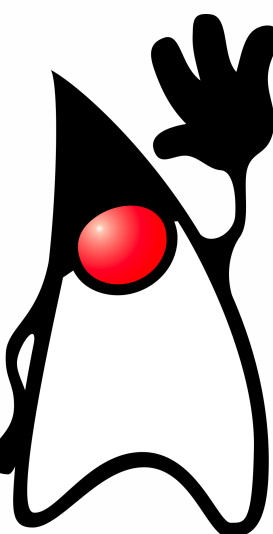


Свойство – Credentials:

Подтверждает аутентичность пользователя

В терминах логин-пароль – это пароль

Authorities – это права которые зарегистрированы:



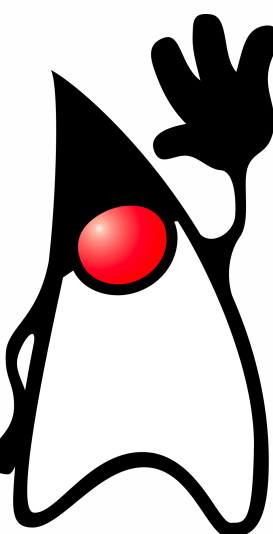
Это один из flow (обычно многое другое)

Пользователь отправляет логин-пароль (или много другое)

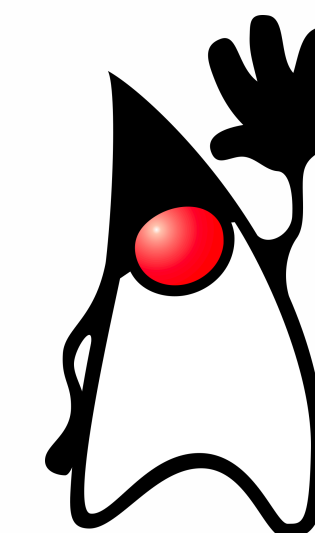
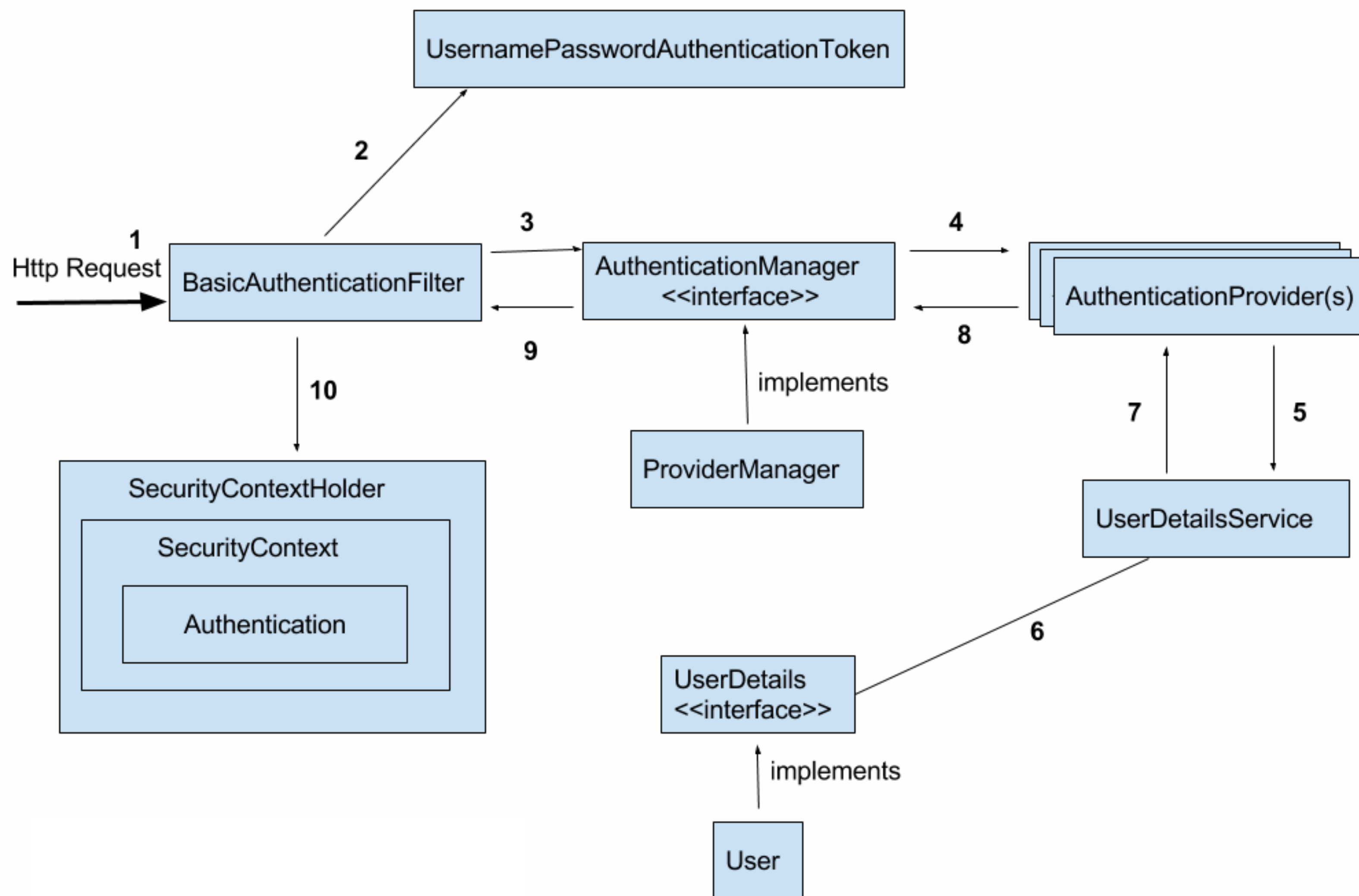
Spring Security заполняет SecurityContextHolder объектом Authentication сначала ненастоящим (логин-пароль)

Spring Security заполняет SecurityContextHolder настоящим объектом Authentication

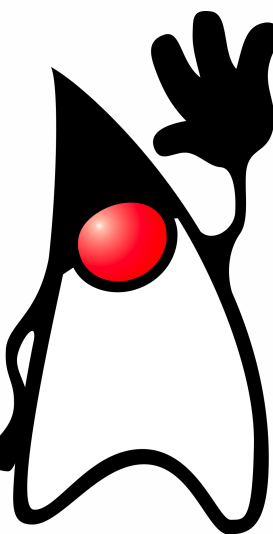
Приложение использует Authentication из SecurityContext



Как происходит работа



UserDetails – абстрактное представление учетной записи пользователя

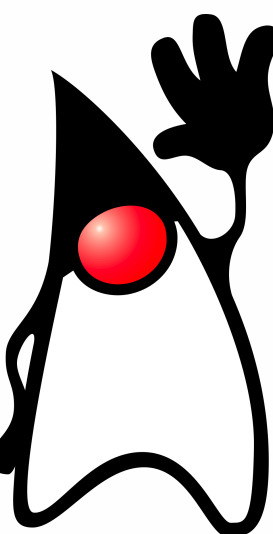


UserDetailsService – интерфейс объекта, реализующего загрузку пользовательских данных из хранилища

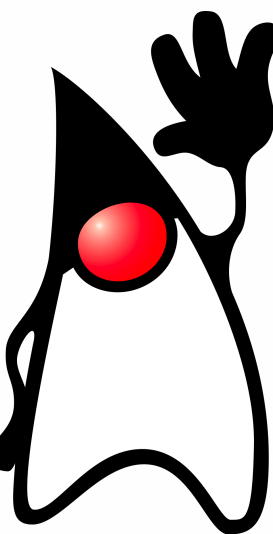
Реализации:

InMemoryDaoImpl

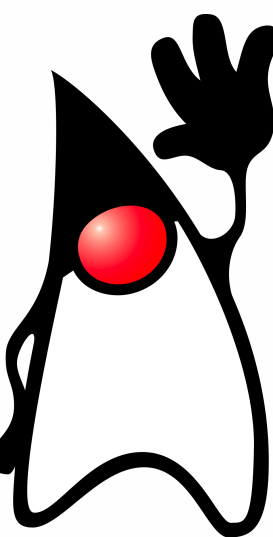
JdbcDaoImpl



```
@Bean
public UserDetailsService daoUserDetailsService(
    DataSource dataSource
) {
    JdbcDaoImpl dao = new JdbcDaoImpl();
    dao.setDataSource(dataSource);
    return dao;
}
```



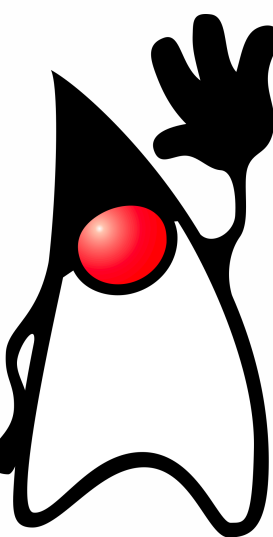
AuthenticationManager – интерфейс объекта выполняющего аутентификацию



AuthProvider – интерфейс объекта, выполняющего аутентификацию

Масса реализаций уже готовых

Скорее всего будет залазить в UserDetails



PasswordEncoder :

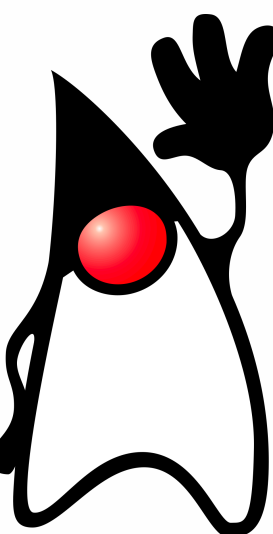
ShaPasswordEncoder

Md5PasswordEncoder

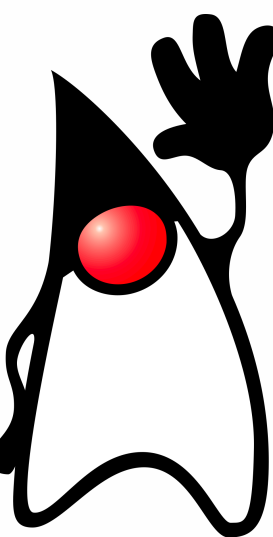
Md4PasswordEncoder

PlaintextPasswordEncoder

Какой из не Deprecated?



AccessDecisionManager – интерфейс объекта, который принимает решение о доступе к запрашиваемому ресурсу

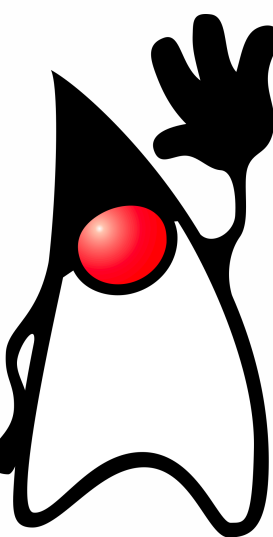


AbstractSecurityInterceptor:

Authentication из SecurityContextHolder

AuthenticationManager

AccessDecisionManager



FilterSecurityInterceptor

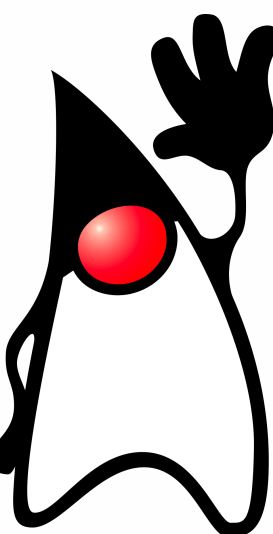
Управляет доступом на уровне URL

Использует цепочку HTTP фильтров для управления доступом

MethodSecurityInterceptor

Управляет доступом на уровне методов класса

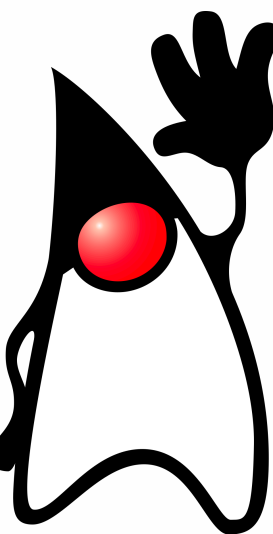
Использует Spring AOP




```
@EnableWebSecurity  
@Configuration  
public class ConfigClass extends WebSecurityConfigurerAdapter {}
```

ИЛИ

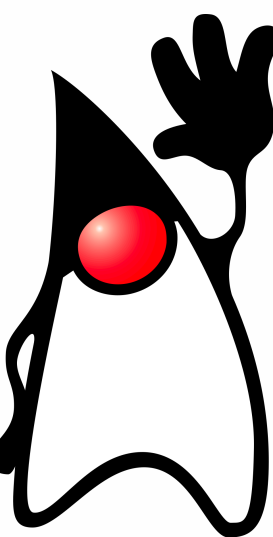
```
@EnableWebSecurity  
@SpringBootApplication  
public class App {}
```



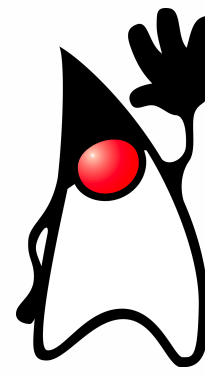
При написании сложных приложений не всегда можно обойтись namespace-based или Java-based подходами

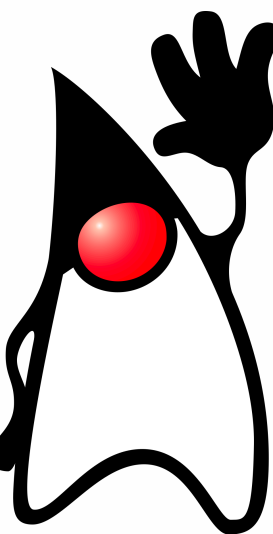
Для тонкой настройки приходится использовать классический подход

Их можно использовать одновременно в конфигурации приложения (зачастую так и бывает)



Ваши вопросы?

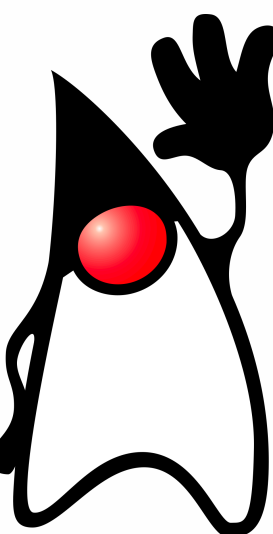
- Задачи безопасности
- Архитектура Spring Security
- • **Spring Security и Web**
- Аутентификация
- Авторизация по URL

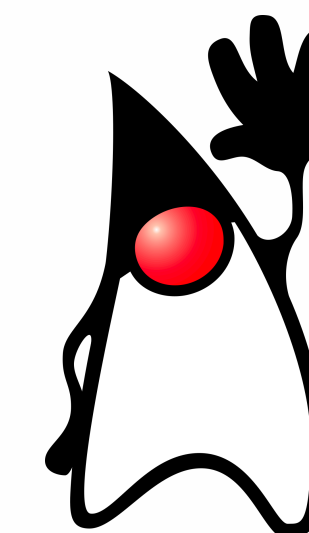
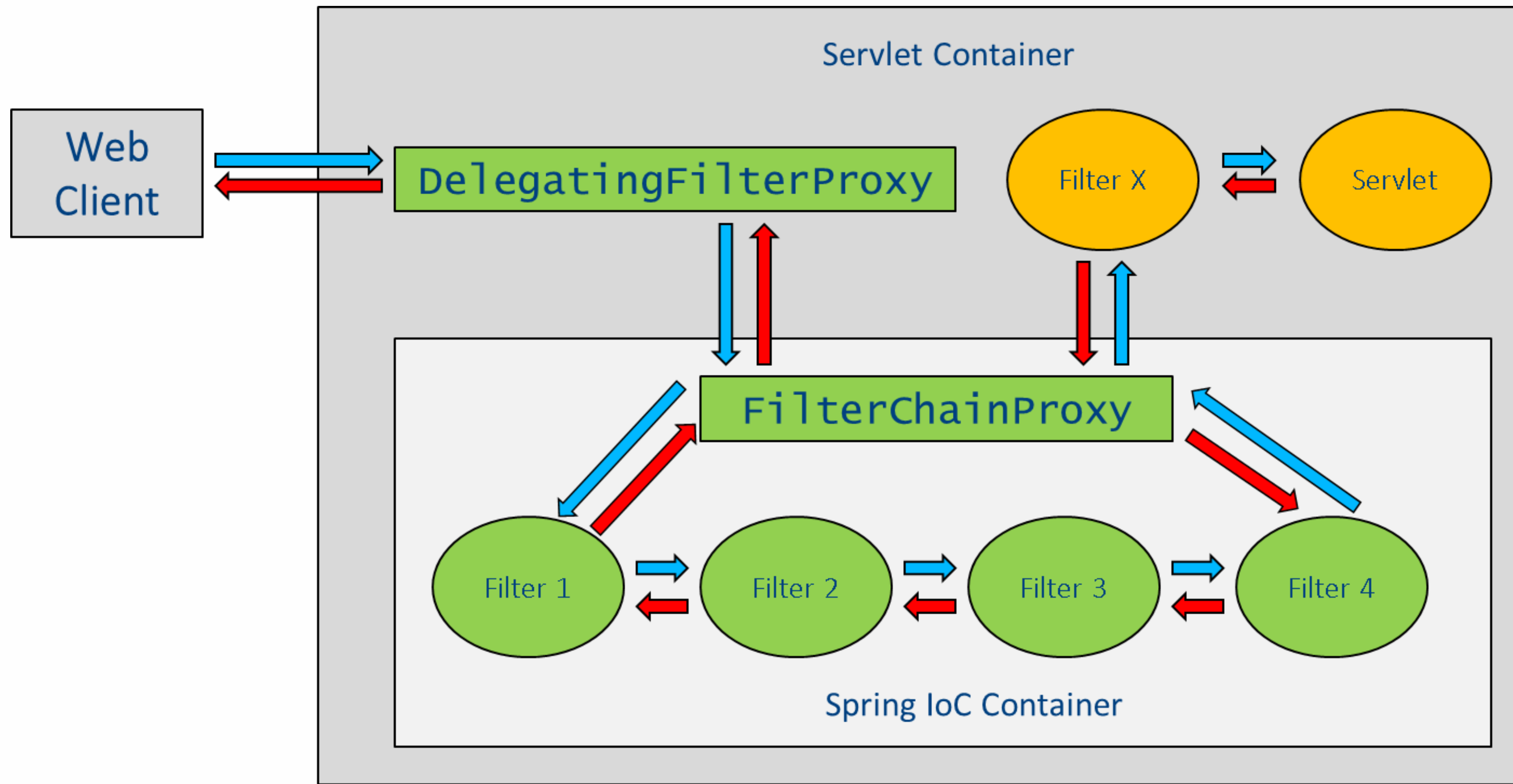


03

Spring Security и Web

- Интеграция Spring Security со средой Web достигается через использование Servlet фильтров
- Фильтры объединены в цепочки
- Каждый фильтр реализует какой-то аспект механизма безопасности
- Важна последовательность фильтров в цепочке

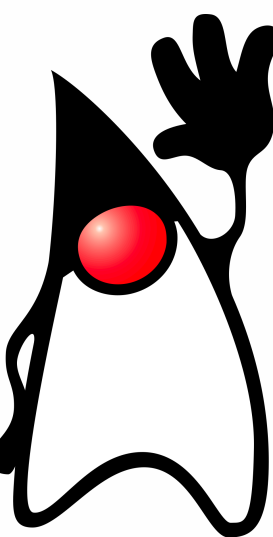




DelegatingFilterProxy

Интегрирует цепочки фильтров в последовательность обработки HTTP запроса

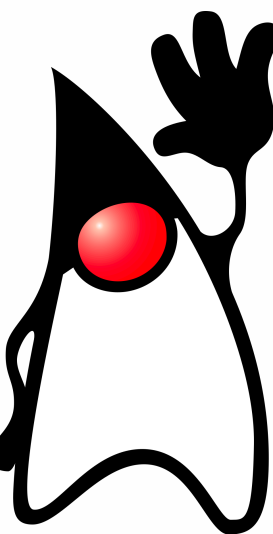
Единственная задача – вызов цепочки фильтров Spring Security




```
@EnableWebSecurity  
@Configuration  
public class ConfigClass extends  
    WebSecurityConfigurerAdapter {}
```

или

```
@EnableWebSecurity  
@SpringBootApplication  
public class App {}
```



@EnableWebSecurity

@Configuration

```
public class SpringSecurityConfig extends  
    WebSecurityConfigurerAdapter {
```

@Override

```
protected void configure(HttpSecurity http) throws Exception {
```

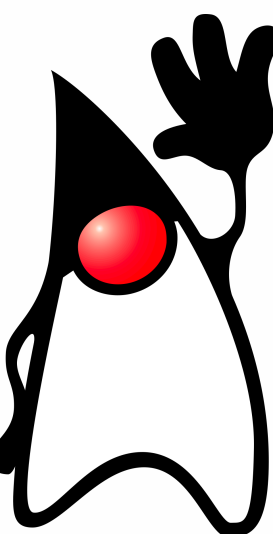
```
    http
```

```
        .csrf().disable()
```

```
        .authorizeRequests();
```

```
    }
```

```
}
```



Фильтры Spring Security реализует конкретные аспекты безопасности:

Совсем служебные

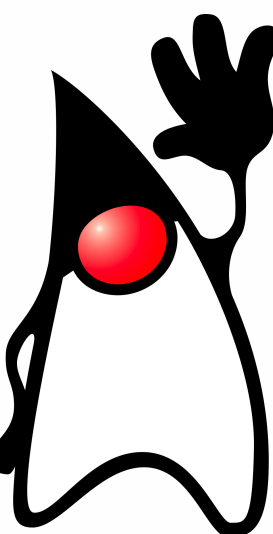
Аутентификация

Авторизация

Обработка ошибок

Цепочка фильтров:

В случае Java- based конфигурации создаётся неявно



ChannelProcessingFilter

ConcurrentSessionFilter

SecurityContextPersistenceFilter

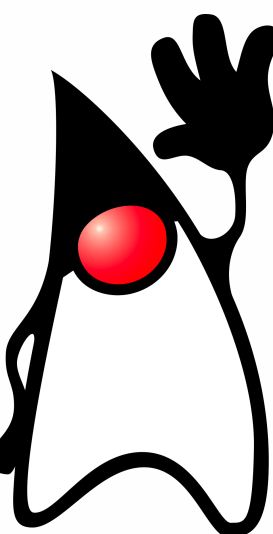
Фильтр(ы) аутентификации

RememberMeAuthenticationFilter

AnonymousAuthenticationFilter

ExceptionTranslationFilter

FilterSecurityInterceptor



ChannelProcessingFilter – если по HTTP, то перенаправляет на HTTPS

ConcurrentSessionFilter – защита от повторяющихся сессий

SecurityContextPersistenceFilter – создаёт Security-контекст

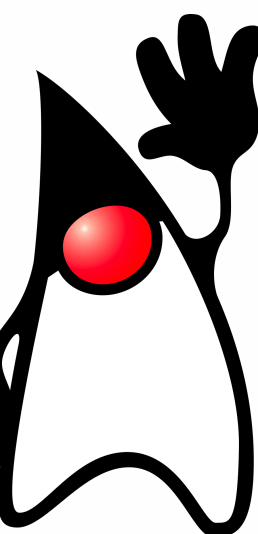
Фильтр(ы) аутентификации – их тема, пройдем на другом занятии

RememberMeAuthenticationFilter – вторичная аутентификация
«Запомни меня»

AnonymousAuthenticationFilter – тоже вторичная аутентификация

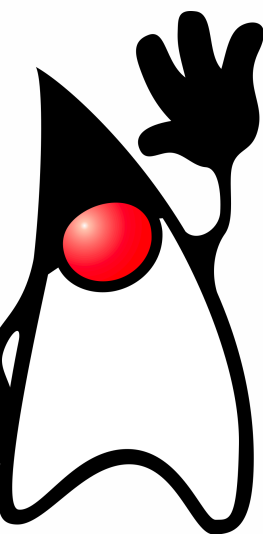
ExceptionTranslationFilter – обрабатывает исключения

FilterSecurityInterceptor – а вот он как раз занимается безопасностью



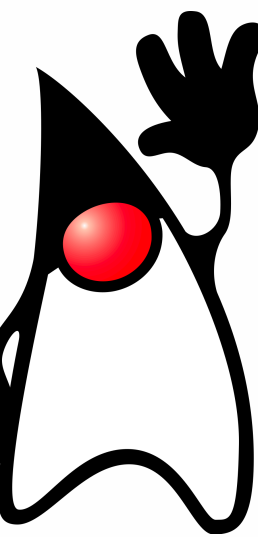
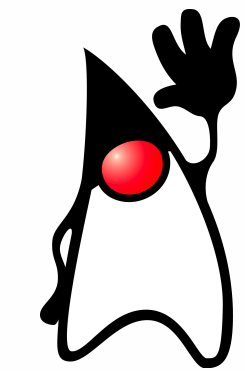
Ваши вопросы?

Упражнение №2 в действии



**Ваши вопросы по
примеру?**

- Задачи безопасности
- Архитектура Spring Security
- Spring Security и Web
- **Аутентификация**
- Авторизация по URL

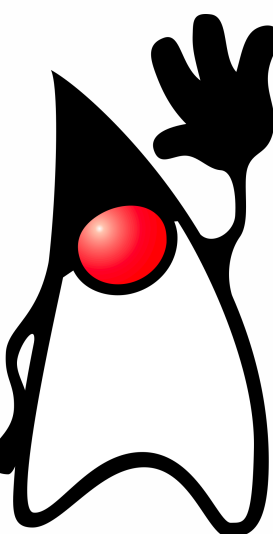


04

Аутентификация

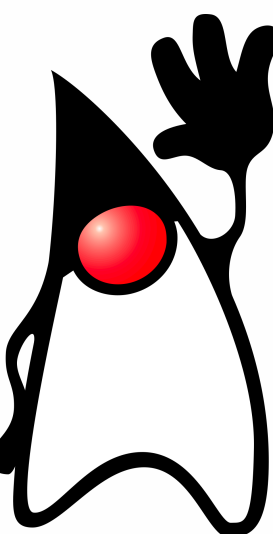
Сценарий аутентификации в Web:

- Клиент переходит по ссылке на фотку в контакте
- Запрос уходит на сервер, сервер решает, что незарегистрированный пользователь не может смотреть
- Сервер отправляет ответ, что нужна аутентификация



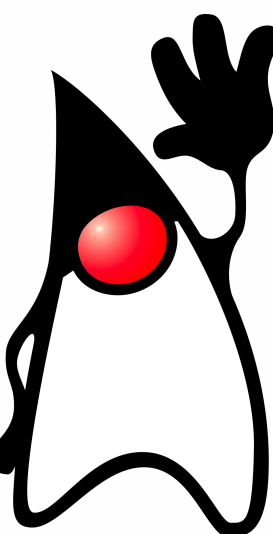
Сценарий аутентификации в Web (продолжение):

- Браузер в зависимости от механизма аутентификации:
 - переходит на Login Page
 - или другим способом собирает информацию о клиенте (BASIC Auth, X.509)
- Форма отправляет данные на сервер через HTTP тело или HTTP заголовки



Сценарий аутентификации в Web (продолжение):

- Сервер проверят Credentials:
- Сервер направляет клиента на первоначальную страницу в контакте.



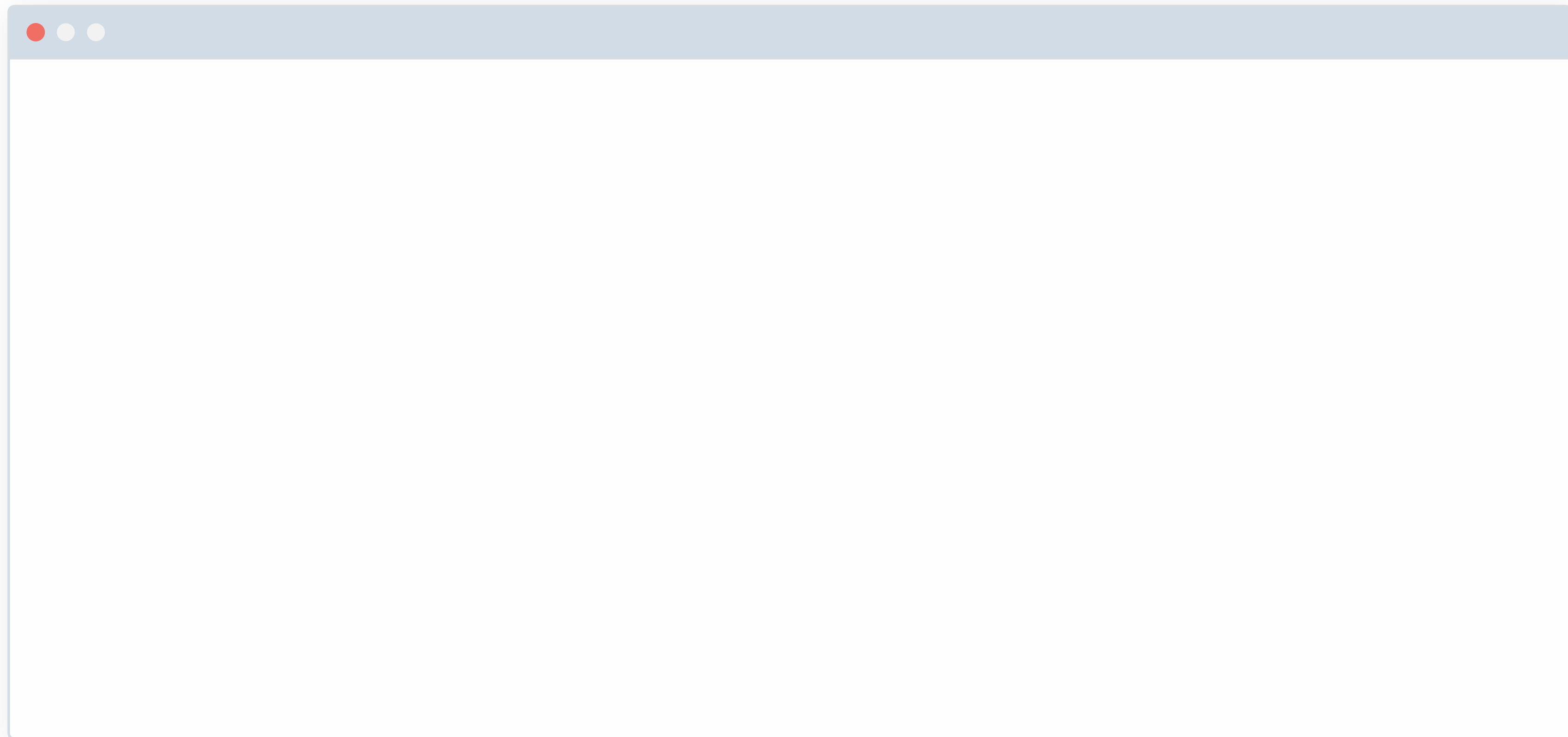
04.1

Form-based Authentication

- UsernamePasswordAuthenticationFilter – на сервер
- На клиенте реализуется web формой, содержащей поля:
 - Имя пользователя
 - Пароль
- Эта форма уже есть стандартная, а можете свою написать

```
@Override
public void configure(HttpSecurity http) throws Exception {
    http
        .formLogin()
        .loginProcessingUrl("/j_spring_security_check")
        .usernameParameter("j_username")
        .passwordParameter("j_password")
        .loginPage("/public/login.jsp");
}
```


Смотрим Form-based аутентификацию

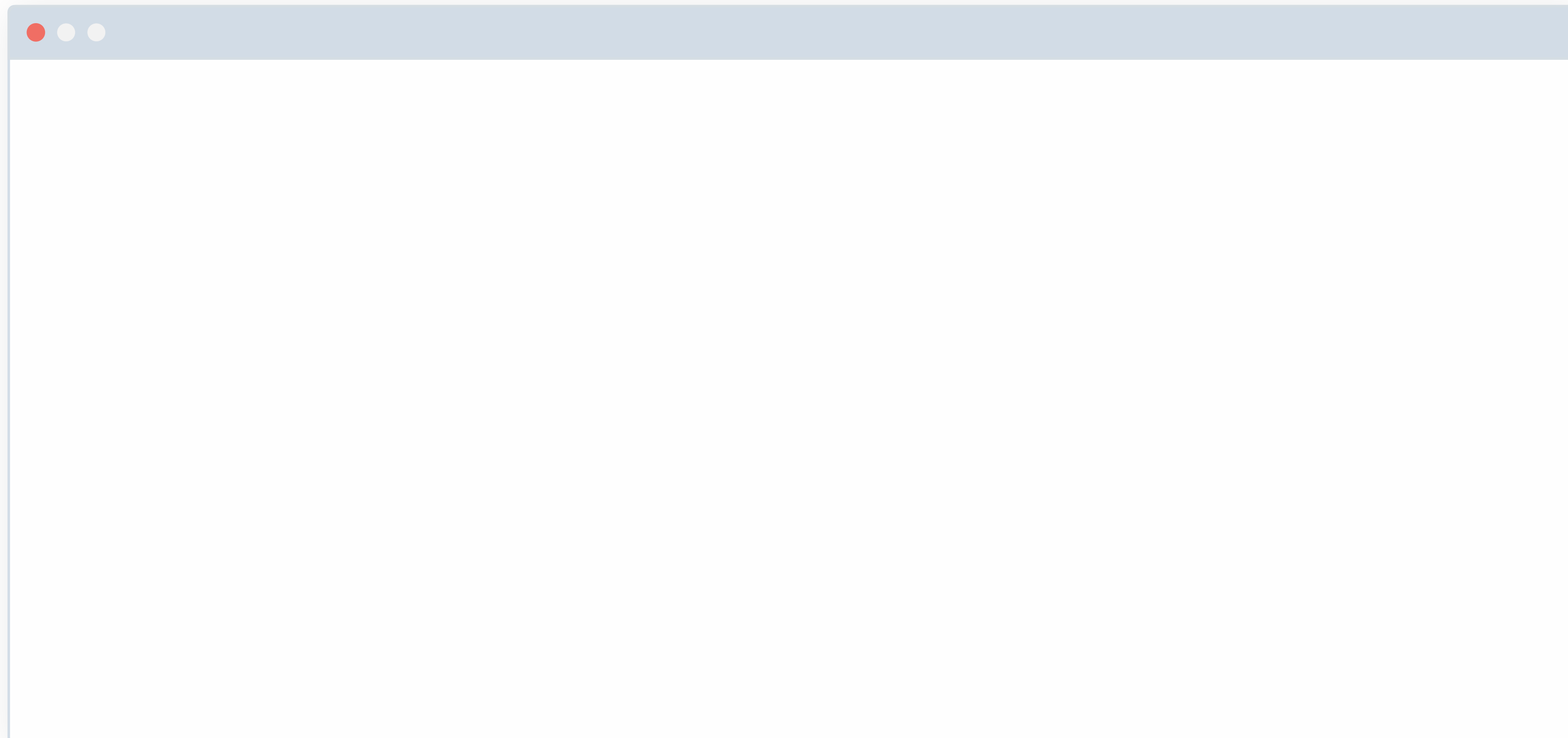


Пара слов:

- Логин форма настраивается
- Вы можете использовать свою
- Стратегии (куда перенаправлять – тоже настраиваются)
- Это одна из самых гибких частей)
- Если не хочется использовать готовую – то можно свой метод (принимающий запрос). Вполне реально пишется

```
1 //
2
3 @PostMapping("/auth")
4 public AuthenticationResponseDto authenticate(
5     @ApiParam(name = "Тело запроса", value = "Запрос на аутентификацию", required = true)
6     @RequestBody AuthenticationRequestDto requestDto,
7     HttpServletRequest request, HttpServletResponse response
8 ) {
9     String login = requestDto.getLogin();
10    String password = requestDto.getPassword();
11    Authentication authRequest = new UsernamePasswordAuthenticationToken(login, password);
12    Authentication auth = authenticationManager.authenticate(authRequest);
13    UserDetailsAdapter userDetails = (UserDetailsAdapter) auth.getPrincipal();
14    rememberMeServices.loginSuccess(request, response, auth);
15    return new AuthenticationResponseDto(
16        userDetails.getUsername(), userDetails.getName(),
17    );
18 }
```

- Настроить аутентфикацию через веб-форму, чтобы при логине она перенаправляла на success, если не было страницы изначально
- * URL и страницу при failure



Ваши вопросы?

04.2

HTTP Basic Authentication

- HTTP Basic Access аутентификация – это способ представления имени пользователя и пароля браузером или иным HTTP клиентом
- Является стандартной для протокола HTTP
- Определена в RFC-1945 (Hypertext Transfer Protocol – HTTP/1.0)
- Суть – передача закодированных в Base64 имени/пароля в виде HTTP заголовка Authorization

Преимущества:

- Простота механизма
- Поддержка всеми браузерами
- Идеально для микросервисов

Недостатки:

- Имя пользователя и пароль передаются в открытом виде (необходимо использовать HTTPS)

- BasicAuthenticationFilter
- BasicAuthenticationFilter читает из заголовка HTTP запроса имя и пароль и использует для аутентификации
- BasicAuthenticationFilter соответствует RFC-1945

```
@Override
public void configure(HttpSecurity http)
    throws Exception {

    http.httpBasic();
}

// в spring-boot-starter-web-security
// это уже есть
```

Ваши вопросы?

Прикрутить HTTP-basic аутентификацию

```
git clone https://github.com/kataus/spring-framework-20
```

Ваши вопросы?

04.3

Anonymous Authentication

- Есть открытые ресурсы
- Часто такие ресурсы исключаются из обработки цепочкой фильтров
- В результате не для всех запросов подсистема Spring Security будет не доступен заполненный SecurityContext
- А иногда код использует SecurityContext
- И получаются exceptions

- Anonymous Authentication – это типа паттерн NullObject
- (уже не Optional, скорее NaN в мире double)
- Основная идея – иметь пользователя с самымидохлыми правами
- Такое есть в FTP, Гость в ОС и Chrome и т.д.

- Теперь можно ограничить доступ просто
- Иметь полностью доступную инфраструктуру Spring Security для всех запросов к системе

AnonymousAuthenticationToken

- Реализация интерфейса Authentication для анонимной аутентификации
- Хранит список GrantedAuthority для анонимной аутентификации

AnonymousAuthenticationFilter

- Аутентифицирует пользователя как анонимного в случае если отсутствует другая аутентификация
- Находится в цепочке после всех настроенных фильтров аутентификации

```
@Override
protected void configure(HttpSecurity http)
{
    http.httpBasic().and()
        .anonymous()
        .authorities("ROLE_ANONYMOUS")
        .principal("anonymous");
}
```

Ваши вопросы?

Anonymous аутентификация (/public), anonymous – логин для анонима

* Поправьте код, выводящий логин из /authenticated (не в контроллере)



```
git clone https://github.com/kataus/spring-framework-20
```

Ваши вопросы?

04.4

Remember-Me Authentication

- Remember Me аутентификация позволяет сохранять информацию о пользователе между HTTP сессиями
- Использование Remember Me аутентификации позволяет при очередном обращении к приложению использовать ранее введённые имя и пароль
- Обычно реализуется через сохранение cookie в браузере
- Работает только с form-based аутентификацией

- Simple Hash-Based Token
подход (кстати переписываем
обычно на JWT)
- Persistent Token подход

После удачной аутентификации сохраняется cookie

Cookie содержит закодированную в Base64 строку с:

- Именем пользователя
- Временем экспирации токена
- Md5 от имени + пароль + время экспирации + ключ

```
base64 (
  username + ":" + expirationTime + ":" +
  md5Hex (
    username + ":" +
    expirationTime + ":" + password
    + ":" + key
  )
)
```

```
base64 (
  username + ":" + expirationTime + ":" +
  md5Hex (
    username + ":" +
    expirationTime + ":" + password
    + ":" + key
  )
)
```

```
@Override
protected void configure (HttpSecurity
http) {
    http.rememberMe ()
        .key ("myAppKey")
        .tokenValiditySeconds (60) ;
}
```

- Альтернатива подходу Simple Hash-Based Token
- Основное отличие - содержимое Remember-Me cookie
- Для работы необходимо хранилище для Remember-Me токенов
- Токены в хранилище доступны по суррогатным ключам (значения из cookie)

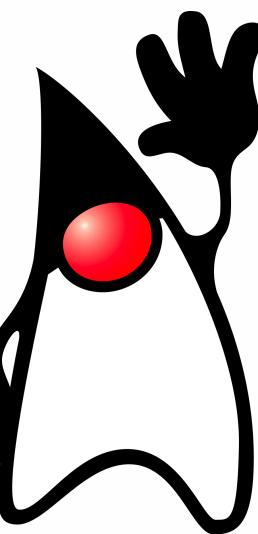
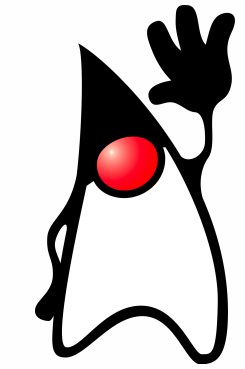
- Хранить выданные токены иногда совсем затратно
- И не хочется лазить в базу за каждым запросом
- Мы лично использовали JWT (аналог `simple-hash-based`) чтобы проверять прямо в сервисах
- Но это не решает проблему логauta – поэтому хранили отозванные токены (пока не истечёт их реальный срок)

04.5

Unit-тестирование

Что сможем после занятия

- Задачи безопасности
- Архитектура Spring Security
- Spring Security и Web
- Аутентификация
- **Авторизация по URL**



05

Авторизация по URL

- ChannelProcessingFilter
- ConcurrentSessionFilter
- SecurityContextPersistenceFilter
- Фильтр(ы) аутентификации
- RememberMeAuthenticationFilter
- AnonymousAuthenticationFilter
- ExceptionTranslationFilter
- **FilterSecurityInterceptor – вот здесь всё**

05.1

Ролевая модель

- Роли представляют собой строчки (хотя есть возможность сделать и сложный объект)
- Когда пользователь вошёл, и где-то в БД они были записаны и вернулись в User Details (смотрим в IDEA), то роли теперь называются Granted Authority (так и называется интерфейс)
- По идее они динамические и записав в базу роль мы добавим права пользователя до следующей аутентификации.

Роли (строчки) в приложении называют так:

- ROLE_ADMIN, ROLE_ANONYMOUS, ROLE_USER
- ADMIN, USER... (вот такое пишется в некоторых служебных классах, ROLE_ добавляется автоматически)
- ROLE_CAN_PACK_BOX – в крутых приложениях

Смотрим какие бывают роли

```
hasRole("ADMIN") эквивалентно hasAuthority("ROLE_ADMIN")
```


- Роли в Spring Security (только для ролевой модели) – Flat, т.е. не иерархичные и смысл мы в них вкладываем какой хотим (собственно строки)
- Т.е. у Вас могут быть ROLE_ANONYMOUS, ROLE_USER, ROLE_ADMIN, но для Spring Security они вообще независимы.
- А какой смысл мы в это вкладывает – решаем мы сами.
- Это касается ролевой модели, а вот ACL – как раз наследуется.

```
@Bean
public RoleHierarchyImpl roleHierarchy() {
    RoleHierarchyImpl roleHierarchy = new RoleHierarchyImpl();
    roleHierarchy.setHierarchy("ROLE_ADMIN > ROLE_USER");
    return roleHierarchy;
}
```

Ваши вопросы?

05.2

Авторизация на основе урлов

- Проверка прав осуществляется в `AccessDecisionManager`
- `AccessDecisionManager` при проверке прав опирается на `GrantedAuthoritys`
- Смотрим)

Вот здесь можете как раз реализовать свою ролевую модель в `AccessDecisionManager`

Реализации (это голосовалки))):

- `AffirmativeBased`
- `ConsensusBased`
- `UnanimousBased` - "default"

Они опрашивают `AccessDecisionVoter` для реализации проверки

Придумайте где требуется

- AffirmativeBased
- ConsensusBased
- UnanimousBased
- Ваш Custom Voter

Можно самостоятельно реализовывать логику проверки в виде `AccessDecisionVoter`

Spring Security предлагает готовые реализации:

- `RoleVoter`
- `AuthenticatedVoter`
- `AclEntryVoter`

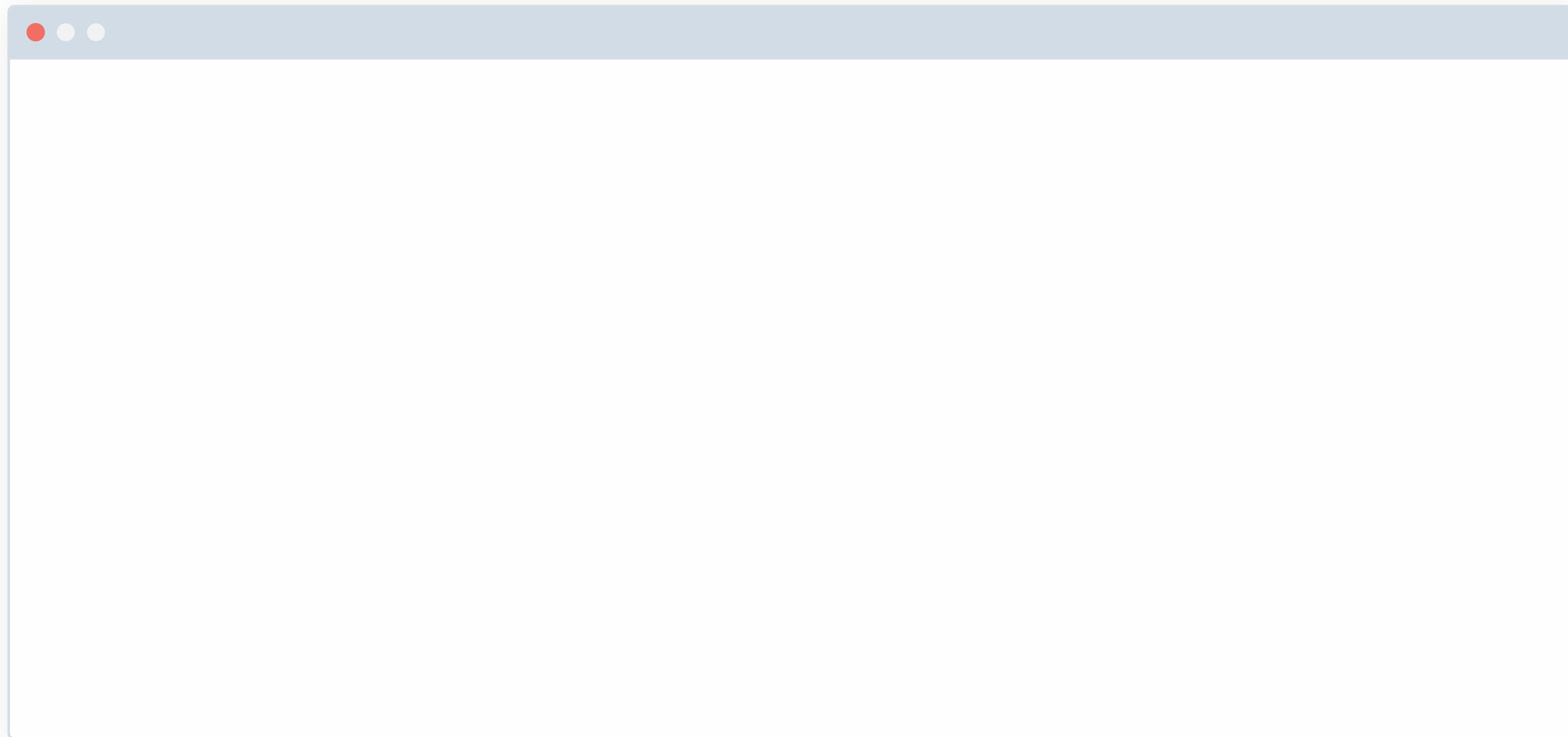
Класс RoleVoter голосует так:

- есть роль – доступ разрешен
- нет роли – доступ запрещен

Класс AuthenticatedVoter выносит вердикт на основе типа аутентификации (обычная, remember-me, anonymous)

- В момент получения управления объектом `FilterSecurityInterceptor` пользователь уже аутентифицирован
- По умолчанию доступ есть у всех на все ресурсы
- Все паттерны ходят «сверху-вниз», срабатывает первый подошедший

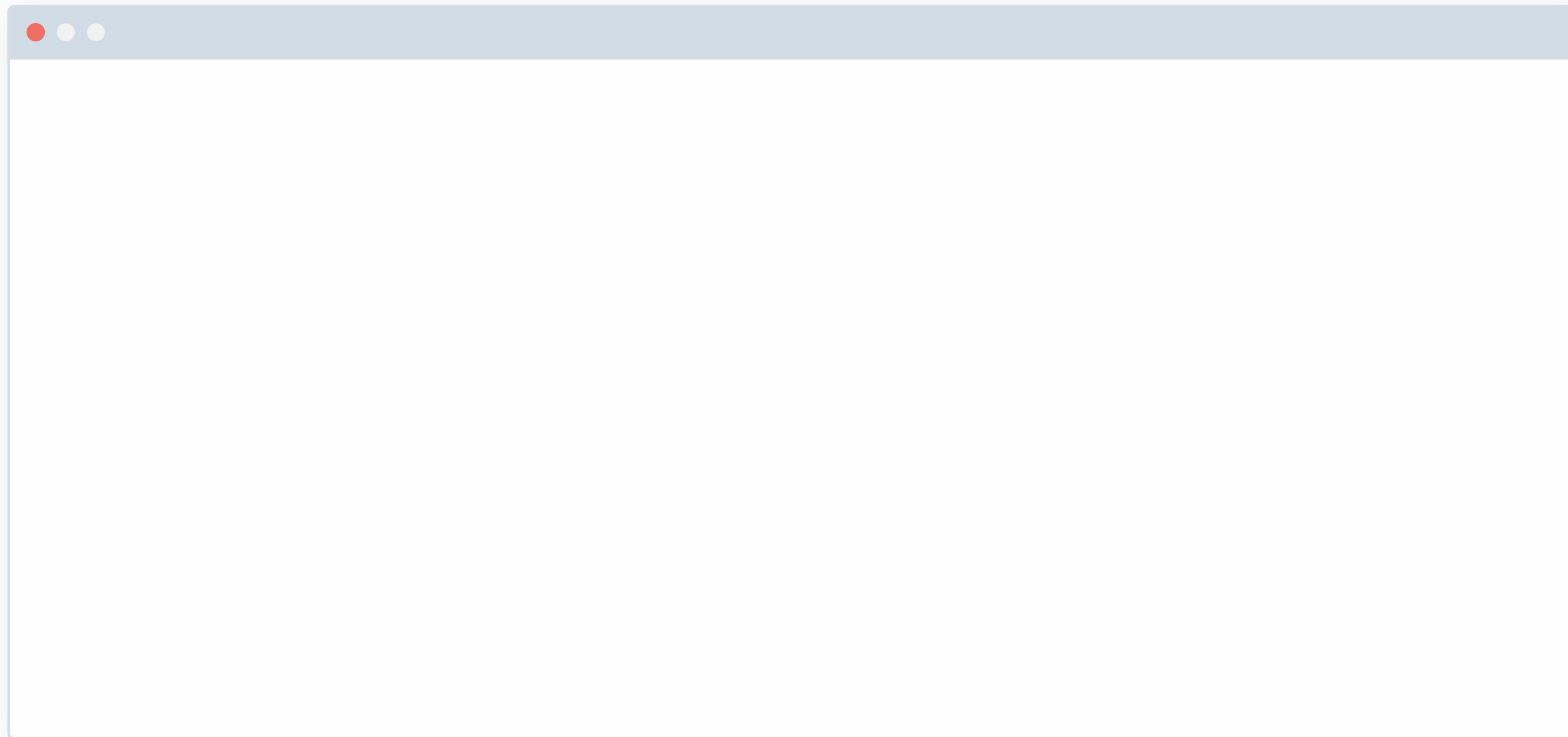
Смотрим настройки авторизации урлов



Ваши вопросы?

Добавить все проверки, что бы они соотносились с ролями и страницами.

Не забудем дать админу доступ к страницам пользователя



Ваши вопросы?

05.3

Авторизация на основе вызовов методов

- MethodSecurityInterceptor
- Это всё AOP – со всеми вытекающими последствиями (класс должен быть в контексте, внутренние обёртки и т.д.)


```
@EnableGlobalMethodSecurity
@EnableWebSecurity
@Configuration
public class MySecurityConfig extends GlobalMethodSecurityConfiguration {
    @Bean
    public MethodSecurityInterceptor methodSecurityInterceptor() {
        return interceptor;
    }
}
```

3 способа

- Аннотацию @Secured – простая
- Аннотации JSR-250
- Spring Security аннотации позволяющие поддерживающие EL выражения (самая круть)

@EnableGlobalMethodSecurity есть атрибуты:

- pre-post-annotations
- jsr250-annotations
- secured-annotations

Все отключены по умолчанию.

Spring Security аннотации, поддерживающие EL:

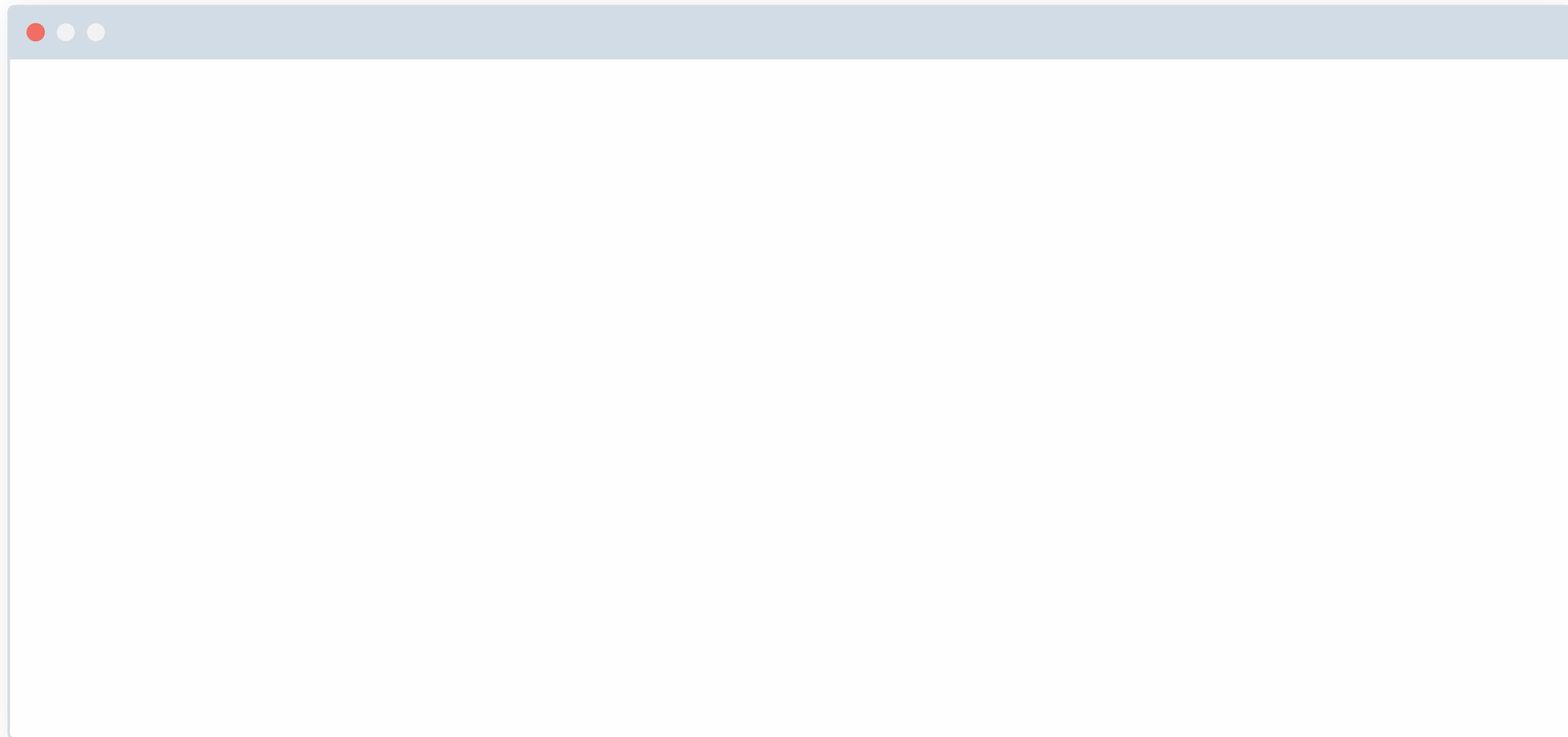
- @PreAuthorize
- @PreFilter
- @PostAuthorize
- @PostFilter

```
@PreAuthorize("hasRole('ROLE_USER')")
public List<News> getNews() {
    ...
}

@Secured({"ROLE_USER", "ROLE_ADMIN"})
public List<News> getNews() {
    ...
}
```

```
@PostFilter("hasPermission(filterObject, 'READ')")
public List<BankAccount> getBankAccounts(Customer customer) {
    ...
}
```

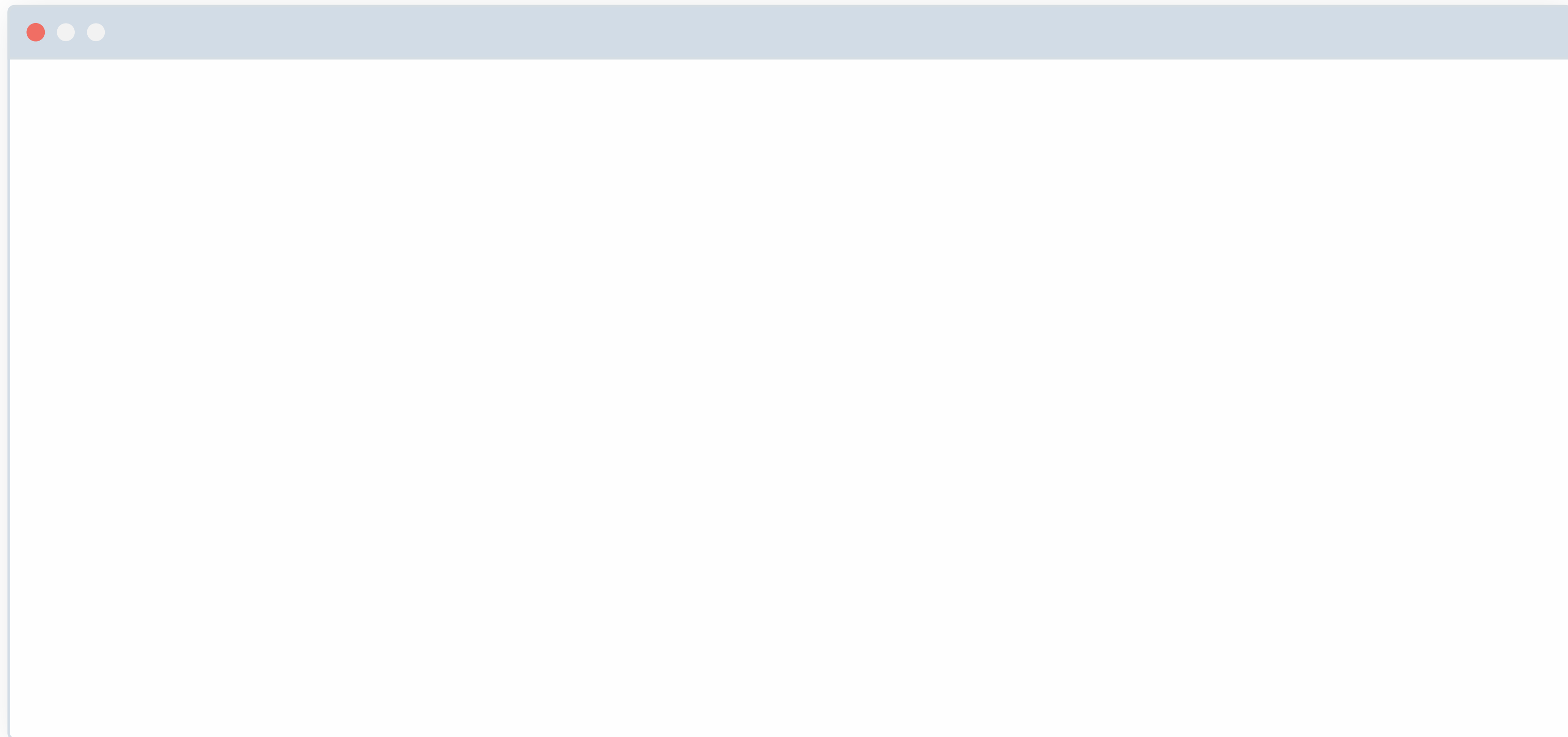
Смотрим авторизацию на основе вызовов методов



Ваши вопросы?

Доделаем авторизацию на уровне урлов

Если хотим – на основе вызова методов



Ваши вопросы?

Спасибо за внимание!

