**Spring security –** фреймворк для обеспечения аутентификации и авторизации пользователей. Легко интегрируется с Servlet API и Web MVC, защищает от атак типа фиксация сессии, кликджекинг, межсайтовая подделка запроса и др.

Аутентификация – кто вы? Авторизация – что вам разрешено делать.

**Основные возможности:**

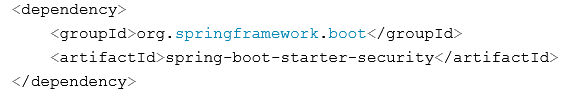
* **Шифрование паролей**
* **Аутентификация In-Memory** – для сохранения информации о пользователях и выполнения аутентификации можно использовать временную базу данных, которая остается в оперативной памяти приложения. (Полезно при разработке и тестировании)
* **LDAP**-**аутентификация** - Lightweight Directory Access Protocol (LDAP) — протокол аутентификации учетных записей пользователей в организациях. Позволяет определять структуру пользователей и групп пользователей, назначать им права доступа.
* **Управление сессией**
* **Remember Me Authentication –** встроенный механизм распознавания, благодаря которому пользователям не нужно вводить учетные данные при каждом посещении сайта.
* **OAuth 2.0 -** открытый стандарт проверки прав пользователя с помощью сервиса авторизации. Он также используется для реализации таких функций, как вход через учетные записи Facebook, Google и других крупных площадок.

В Web MVC Spring Security основана на фильтрах сервлетов. Есть цепочка фильтров, которая может предотвратить доступ к контроллеру.

**FilterChainProxy** – фильтр в цепочке, который отвечает за SpringSecurity. Он установлен по умолчанию, и применяется к каждому запросу. С точки зрения контейнера, SpringSecurity это один фильтр. Но внутри него есть дополнительные фильтры, которым он делегирует работу.

**Подключение Spring Security к проекту:**

Добавить в проект зависимость

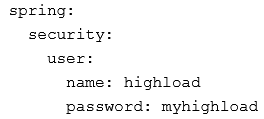


При использовании SpringBoot этого будет достаточно, чтобы добавить базовую аутентификацию. Spring security добавляет своего пользователя с именем user и генерирует ему пароль при запуске. Кроме того:

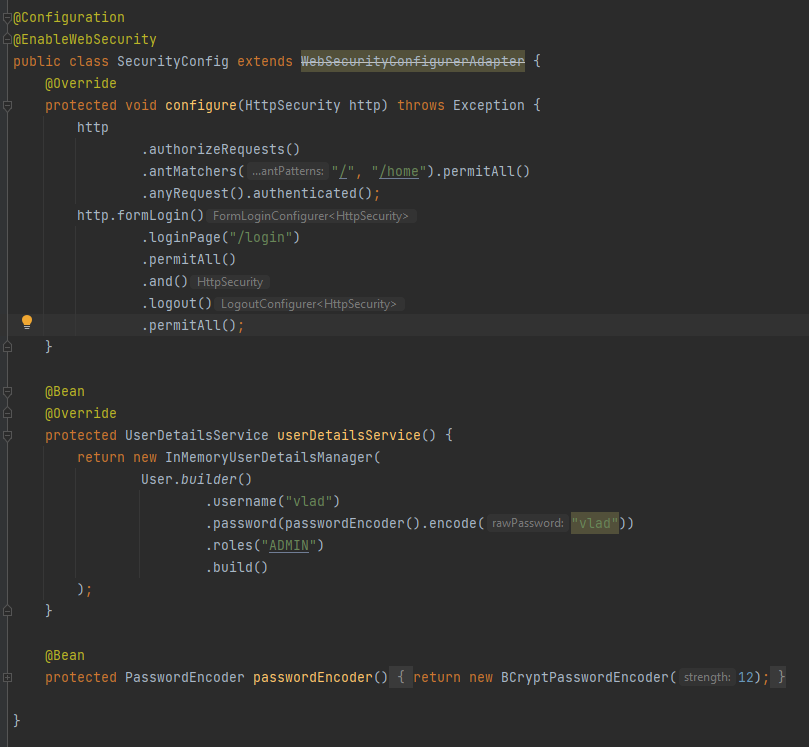
1. Формируется страница с формой для ввода имени и пароля.
2. Сразу начинает работать проверка имени и пароля.
3. Все URL недоступны, пока вы не зайдете в систему.

По умолчанию данные пользователей хранятся в оперативной памяти, пока приложение запущено (In-Memory authentication).

Параметры пользователя по умолчанию можно настроить в *application.yml*



Настроить аутентификацию можно в классе конфигурации. Нужно унаследоваться от класса **WebSecurityConfigurerAdapter** и использовать аннотацию **@EnableWebSecurity.**

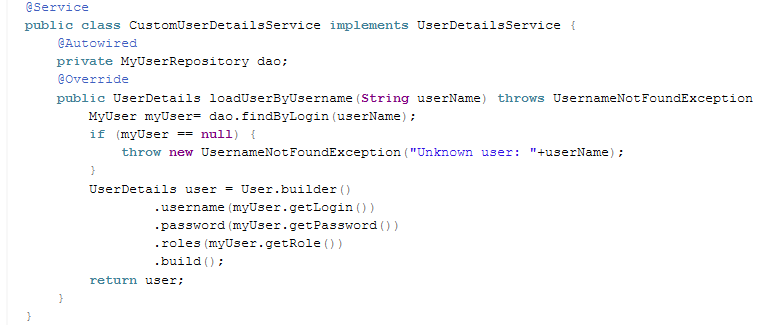


Метод **configure(HttpSecurity)** определяет какие URL пути должны быть защищены, а какие нет.

Используя **AuthenticationManagerBuilder** мы указываем тип аутентификации и добавляем юзеров.

**UserDetailsService –** объект, позволяющий получать пользователя (Объект UserDetails) по имени.

Например можно реализовать получение пользователя из БД.



В интерфейсе UserDetails есть методы для получения хешированного пароля и для получения имени пользователя. Так же есть методы активна или заблокирована учетная запись, истек срок действия учетных данных или какие разрешения есть у пользователей.

**Типы UserDetailsService менеджеров**:

* **InMemoryUserDetailsManager –** пользователи хранятся в оперативной памяти.
* **JdbcUserDetailsManager** – пользователи хранятся в БД. Вы можете настроить его в соответствии с вашей пользовательской структурой таблицы.
* **LDAP –** Пользователи берутся из LDAP. Lightweight Directory Access Protocol, является открытым протоколом, используемым для хранения и получения данных из каталога с иерархической структурой.
* **Кастомная**

В **AuthenticationManager** объявлен один метод **authenticate(Authentication authentication).** Этот метод может вернуть объект Authentication если удалось идентифицировать сотрудника, либо выбросить исключение **AuthenticationException,** если входные данные представляют недействительного принципала. Возвращает null если не может принять решение.

**Authentication –** объект, который предоставляет токен для запроса аутентификации или для принципала который прошел аутентификацию. Также содержит список полномочий, к которым получил доступ принципал.

Как происходит Аутентификация:

В метод **authenticate** **фильтра UsernamePasswordAuthenticationFilter** передается токен с именем и паролем из http заголовка.

1. Фильтр **UsernamePasswordAuthenticationFilter** извлекает имя и пароль пользователя из http заголовка Basic Auth.
2. **AuthenticationManager** в методе **authenticate()** в зависимости от своего типа извлекает с помощью UserDetailsService указанного пользователя (Из БД, из Оперативной памяти и т.п.) в виде объекта UserDetails
3. Пароль из http заголовка хешируется и сравнивается с тем что в объекте UserDetails. Если аутентификация прошла, возвращается новый объект Authentication.

Объект Authentication сохраняется в SecurityContext, а тот в свою очередь в SecurityContextHolder. Получить текущего пользователя можно следующим образом:

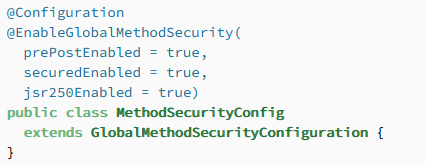


Если включены сессии, то после аутентификации клиенту отправляется уникальный **JSESSIONID.** Клиент отправляет его на сервер во всех последующих запросах. По JSeSSIONID восстанавливается сессия, из нее берется SecurityContext, а из него Authentication.

**PasswordEncoder –** объект определяющий функцию хеширования пароля. Как правило создается бином в конфигурации.

**Method Security**

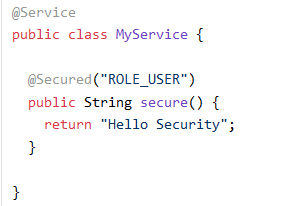
Можно применять правила доступа к выполнению методов Java. Чтобы включить ее используется аннотация **@EnableGlobalMethodSecurity(securedEnabled = true)**



Свойства

* **prepostEnabled** включает аннотации **@PreAuthorize** и **@PostAuthorize**.
* **securedEnabled** включчает аннотацию **@Secured.**
* **Jsr250Enabled** включает аннотацию **@RoleAllowed.**

Аннотация **@Secured(“ROLE\_USER”)** указывает, что данный метод может выполнить только пользователь с ролью USER. Если доступ к методу запрещен, бросается исключение **AccessDeniedException.** Данная аннотация не поддерживает SpEL.

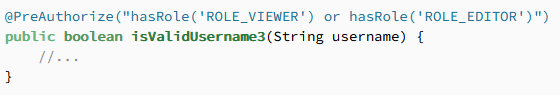


Аннотация **@RolesAllowed -** это аннотация JSR-250, эквивалентная аннотации @Secured.

Аннотации @PreAuthorize и @PostAuthorize обеспечивают контроль доступа **на основе выражений**, предикаты могут быть написаны с использованием SpEL.

Аннотация **@PreAuthorize** проверяет заданное выражение **перед входом в метод**, тогда как аннотация **@PostAuthorize** проверяет его **после** выполнения метода и **может изменить результат**.

Например Аннотацию @RolesAllowed можно переписать с помощью @PreAuthorize следующим образом:



Более того, мы можем **использовать аргументы метода** внутри выражения. Например следующий метод будет вызван если параметр совпадает с именем текущего пользователя:

