**Введение:**

Внутри SequrityFilterChane содержится ряд фильтров для аутентификации, один из них UsernamePasswordAuthenticationFilter, который с помощью AuthenticationManager может подгрузить разные провайдеры, и с помощью этих провайдеров и происходит процесс аутентификации. То есть, AuthenticationManager проходится циклом по всем провайдерам и по очереди пытается произвести аутентификацию.

Как только аутентификация будет пройдена, значит пользователь есть в нашей системе. После того, как пользователь прошел процесс аутентификацию, все начинается крутиться вокруг SequrityContext, сам же SequrityContext является оберткой вокруг объекта аутентификации.

UsernamePasswordAuthenticationFilter – реализация происходит через форму логина.

**В нашем приложении может быть настроено несколько провайдеров.**

Список провайдеров:

DaoAuthenticatinProvider:

JWT provider:

OAuth 2 provider:

TestAuthenticatinProvider:

**Объект аутентификации:**

Объект аутентификации — это тот объект, который хранит всю информацию о пользователе, и имеет 3 значения:

1. Principal - тут храниться вся информация о нашем пользователи, по сути, в Principal хранится наше DTO. Тут как правило, наш service должен реализовать интерфейс UserDetailsService.
2. Credentials - как правило, тут хранится пароль, но после успешной аутентификации, те значения, которые хранит в себе Credentials будут удаляться, то есть, все пароли будут удалены, это делается из-за безопасности. Пример: после того как user прошел аутентификацию, чтобы его пароль не гулял по всему нашему приложению, нам нужно почистить Credentials.
3. Authorities - тут храниться роли пользователей и эти роли должны будут реализовать интерфейс GrantedAuthority

Чтобы иметь доступ к SequrityContext, нужно использовать SequrityContexHolder. SequrityContexHolder имеет много стратегий и одна из самых распространенных это ThreadLocal<>.

ThreadLocal<> — это что-то типа ассативного массива, где ключом является поток, а значением параметризованное значение в нашем случаи это SequrityContext.

После удачной аутентификации мы положим данные пользователя в ThreadLocal<SequrityContext>, а когда пользователь выйдет, нам нужно будет почистить ThreadLocal<SequrityContext>.