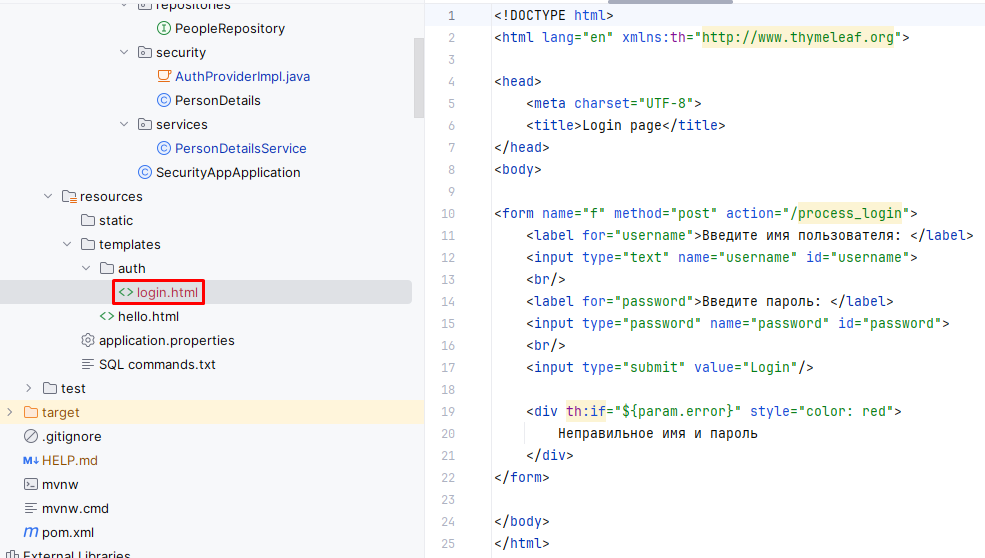
**Spring Security Кастомная страница аутентификации**

**1.** Создадим новый контроллер AuthController. В методе этого контроллера будем просто отдавать представление где будет находится наша форма для аутентификации.

  
  
  
  
  
  
  
  
  
  
  
**2.** Опишем нашу форму для логина в templates/auth/login.html.

Здесь важно учесть, что название полей должны быть именно **username** и **password.** Иначе работать не будет так как spring ожидает именно эти названия полей.



**3.** Теперь перейдем в конфигурацию и скажем Spring Security, что у нас будет своя страница аутентификации. Для этого нам нужно переопределить еще один метод

***public SecurityFilterChain filterChain (httpSecurity http)***. В этом методе будем конфигурировать сам Spring Security, тоесть, какая у нас страничка отвечает за вход, какая страничка отвечает за ошибки и т. д. И помимо этого в этом же методе будем конфигурировать и авторизацию.

Конфигурация происходит по цепочке. Тоесть на объекте HttpSecurity мы по цепочке вызываем нужные методы. Описание методов:

* ***.csrf(AbstractHttpConfigurer::Disable)*** - отключаем защиту от межсайтовой подделки запросов.
* ***.autherizeHttpRequests()*** - когда вызываем этот метод, то теперь все запросы которые приходят к нам в приложении — они будут проходить через нашу авторизацию.
* ***.requestMatcher(«/auth/login», «/error»)*** - указываем на какие адреса пользователь может переходить в не аутентифицированном состоянии. Например, если пользователь перешел по адресу «/auth/login», то мы его должны впускать, тоесть не аутентифицированного пользователя мы должны пускать на эту страничку потому-что он должен иметь доступ к этой страничке, чтобы аутентифицироваться. И так же нужно пускать пользователя на страницу с ошибкой.
* ***permitAll()*** - говорит, что все запросы на страницы которые указаны в методе ***antMathcer()*** мы пропускаем.
* ***anyRequest().authenticated()*** - указываем, что для всех других запросов, которые не находятся в методе ***antMathcer()*** пользователь должен быть аутентифицирован.
* ***and()*** - служит для соединения разных блоков отличающихся по логике. Например, блоки соединяет разные блоки, блок авторизации и блок аутентификации.
* ***formLogin() -*** указывает спрингу, что мы хотим использовать свою страницу для аутентификации.
* ***loginPage(«/auth/login») -*** здесь указывает адрес нашей страницы.
* ***login ProcessingUrl(“/process\_login”)*** - в аругменты мы передаем адрес куда мы хотим отправлять данные с формы. Тоесть мы не будем реализовывать в нашем контроллере еще один метод для приема данных с этой формы, мы не будем реализовать саму аутентификацию. Мы Спрингу даём только **PersonDetailsService**, а всё остальное Spring Security делает сам. И так как мы хотим свою страницу для логина нам нужно только создать страницу, но при этом нам не нужно самим обрабатывать данные с этой формы, мы просто хотим чтобы, данные с формы пришли по определенному адресу «**/process\_login**» (это может быть любой адрес). Другими словами в аругментах метода ***login ProcessingUrl()*** указваем адрес для обработки (адес может быть любой) и в форме аутентификации указываем такой же адрес.
* ***defaultSeccessUrl(«/hello», true)*** - здесь указывается адрес страницы после успешной аутентификации. В качестве второго аругмента нужно передать true если мы хотим. Чтобы он всегда перенаправлял нас после успшной аутентификации.
* ***failureUrl(“/auth/login?error”)*** - здесь указывается адрес на который нужно переадресоваться если аутентификации не пройдет успешно. Здесь укажем, что нужно перенаправиться на ту же самую страничку аутентификации, но еще мы укажем в параметрах запроса ключ **error**. Этот параметр придет в наш контроллер и потом будет передан на представление и в представлении с помощью thymeleaf увидим этот параметр и вывидим сообщение об ошибке.

Все эти методы называются правилами автризации, они читаются сверху вниз, поэтому сначала идут более специфичные правила, а затем идут более общие правила.

Когда запрос приходит в наше приложение, то применяется первый совпавший matcher. Эта механика работает примерно как классический if else. Тоесть если первый if истина, то отсльные if`ы выполняться не будут.

