```
package com.example.springsecurityapplication.config;

//import
com.example.springsecurityapplication.security.AuthenticationProvider;
import
com.example.springsecurityapplication.services.PersonDetailsService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Bean;
import
org.springframework.security.config.annotation.authentication.builders.Au
thenticationManagerBuilder;
import
org.springframework.security.config.annotation.method.configuration.Enabl
eGlobalMethodSecurity;
import
org.springframework.security.config.annotation.web.builders.HttpSecurity;
import
org.springframework.security.config.annotation.web.configuration.EnableWe
bSecurity;
import
org.springframework.security.config.annotation.web.configuration.WebSecur
ityConfiguration;
import
org.springframework.security.config.annotation.web.configuration.WebSecur
ityConfigurerAdapter;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.security.crypto.password.NoOpPasswordEncoder;
import org.springframework.security.crypto.password.PasswordEncoder;

// Основной конфиг для конфигурации безопасности в приложении
@EnableWebSecurity
// Сообщает что в приложении доступно разграничение ролей на основе
аннотаций
//@EnableGlobalMethodSecurity(prePostEnabled = true)
public class SecurityConfig extends WebSecurityConfigurerAdapter {
//    private final AuthenticationProvider authenticationProvider;
//
//    @Autowired
//    public SecurityConfig(AuthenticationProvider
authenticationProvider) {
//        this.authenticationProvider = authenticationProvider;
//    }

    private final PersonDetailsService personDetailsService;

    @Autowired
    public SecurityConfig(PersonDetailsService personDetailsService) {
        this.personDetailsService = personDetailsService;
    }

    // Метод по настройке аутентификации
    protected void configure(AuthenticationManagerBuilder
authenticationManagerBuilder) throws Exception {
        // Производим аутентификацию с помощью сервиса

authenticationManagerBuilder.userDetailsService(personDetailsService)
                .passwordEncoder(getPasswordEncoder());
//
authenticationManagerBuilder.authenticationProvider(authenticationProvide
r);
    }
```

```
    // Конфигурируем сам Spring Security
    // конфигурируем авторизацию
    @Override
    protected void configure(HttpSecurity http) throws Exception{
        // Указываем какой url запрос будет отправляться при заходе на
закрытые страницы

        // Отключаем защиту от межсайтовой подделки запросов
        http
                // Указываем что все страницы должны быть защищены
аутентификации
                .authorizeRequests()
                // Указываем что /admin доступин пользователю с ролью
администратора
//                .antMatchers("/admin").hasAnyRole("ADMIN")
//                .antMatchers("/user").hasAnyRole("USER")
                // Указываем что не аутентифицированные пользователи
могут заходить на страницу с формой аутентификации и на объект ошибки
                // С помощью permitAll указывакем что данные страницы по
умолчанию доступны всем пользователям
                .antMatchers("/auth/login", "/error",
"/auth/registration", "/product", "/product/info/{id}", "/img/**",
"/product/search").permitAll()
                // Указываем что все остальные страницы доступны
пользователю с ролью user и admin
                .anyRequest().hasAnyRole("USER", "ADMIN")
//                // Указываем что для всех остальных страниц необходимо
вызывать метод authenticated, который открываем форму аутентификации
//                .anyRequest().authenticated()
                // Указываем что дальше конфигурироваться будет
аутентификация и соединяем аутентификацию с настройкой доступа
                .and()
                .formLogin().loginPage("/auth/login")
                // Указываем на какой url адрес будут отправляться данные
с формы. Нам уже не нужно будет создавать метод в контроллере и
обрабатывать данные с формы. Мы задали url по умолчанию, который
позволяет обрабатывать форму аутентификации в спринг секьюрити. Спринг
секьюрити будет ждать логин и пароль с формы и затем сверить их с данными
в БД
                .loginProcessingUrl("/process_login")
                // Указываем на какой url необходимо направить
пользователя после успешной аутентификации
                // Вторым аргументом ставим true чтобы перенаправление на
данную страницу шло в любом случае при успешной аутентификации
                .defaultSuccessUrl("/index", true)
                // Указываем куда необходимо перенаправить пользователя
при проваленной аутентификации
                // В url будет передан объект. Данный объект мы будем
проверять на форме и если он есть будет выводить сообщение "Неправильный
логин или пароль"
                .failureUrl("/auth/login")
                .and()
                // Указываем что при переходе на  /logout будет очищена
сессия пользовате и перенаправление на /auth/login

.logout().logoutUrl("/logout").logoutSuccessUrl("/auth/login");
    }

    @Bean
    public PasswordEncoder getPasswordEncoder(){
```

```
        return new BCryptPasswordEncoder();
    }
}
```