Федеральное государственное автономное
образовательное учреждение
высшего образования
«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт космических и информационных технологий
институт

Кафедра «Информатика»
кафедра

ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ № 5

Поддержка архитектуры REST в Spring
Тема

Преподаватель _____ А.С. Черниговский
Подпись, дата Инициалы, Фамилия

Студент КИ19-17/1Б, №031939174 _____ А. К. Никитин
Номер группы, зачетной книжки Подпись, дата Инициалы, Фамилия

Красноярск 2021

1 Цель

Познакомиться с настройкой безопасности в Spring.

2 Задачи

Взять практическое задание №4 и добавить следующий функционал:

1) Добавить простейшую страницу регистрации. Пользователь вводит свои логин и пароль, и данная информация вносится в базу данных, пользователю присваивается роль пользователя (User) приложения.

2) Добавить простейшую форму аутентификации. Форма должна быть создана студентом, а не автоматически сгенерированной Spring.

3) В приложении должен быть предусмотрен пользователь — администратор, с, отличной от роли User, ролью администратора (Admin).

4) Разграничить уровни доступа к страницам приложения. Пользователь (User) имеет доступ только к страницам просмотра всех записей и запросов. Администратор (Admin) имеет возможность добавлять, редактировать и удалять записи.

5) Информация о пользователях и их ролях должна храниться в базе данных. Способ хранения — на усмотрение студента.

6) Предусмотреть возможность пользователю выйти из приложения (logout).

7) Продемонстрировать умение настраивать безопасность на уровне представлений. Для этого реализуйте приветствие пользователя после его входа и отображение элемента на основе его роли.

3 Описание варианта

Класс «Канцтовары».

4 Листинг программы

На листингах 1-5 представлен программный код классов-конфигураторов.

Листинг 1 – Конфигурация БД

```
package ru.nikitin.configs;

import lombok.RequiredArgsConstructor;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;
import org.springframework.context.annotation.PropertySource;
import org.springframework.core.env.Environment;
import org.springframework.data.jpa.repository.config.EnableJpaRepositories;
import org.springframework.jdbc.datasource.DriverManagerDataSource;
import org.springframework.orm.jpa.JpaTransactionManager;
import org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean;
import org.springframework.orm.jpa.vendor.HibernateJpaVendorAdapter;
import org.springframework.transaction.PlatformTransactionManager;

import javax.persistence.EntityManagerFactory;
import javax.sql.DataSource;
import java.util.Objects;
import java.util.Properties;

@Configuration
@ComponentScan("ru.nikitin")
@PropertySource("classpath:application.properties")
@EnableJpaRepositories("ru.nikitin")
@RequiredArgsConstructor
public class AppConfig {
    private final Environment env;

    @Bean
    public DataSource dataSource() {
        DriverManagerDataSource dataSource = new DriverManagerDataSource();

dataSource.setDriverClassName(Objects.requireNonNull(env.getProperty("spring.dat
asource.driverClassName")));
```

Окончание листинга 1

```
        dataSource.setUrl(env.getProperty("spring.dataSource.url"));
        dataSource.setUsername(env.getProperty("spring.dataSource.username"));
        dataSource.setPassword(env.getProperty("spring.dataSource.password"));

        return dataSource;
    }

    @Bean
    public Properties jpaProperties() {
        return new Properties();
    }

    @Bean
    @Autowired
    public EntityManagerFactory entityManagerFactory(DataSource dataSource,
Properties jpaProperties) {
        HibernateJpaVendorAdapter vendorAdapter = new
HibernateJpaVendorAdapter();
        LocalContainerEntityManagerFactoryBean factory = new
LocalContainerEntityManagerFactoryBean();

        factory.setJpaVendorAdapter(vendorAdapter);
        factory.setPackagesToScan("ru.nikitin");
        factory.setDataSource(dataSource);
        factory.setJpaProperties(jpaProperties);
        factory.afterPropertiesSet();
        return factory.getObject();
    }

    @Bean
    @Autowired
    public PlatformTransactionManager transactionManager(EntityManagerFactory
entityManagerFactory) {
        JpaTransactionManager txManager = new JpaTransactionManager();
        txManager.setEntityManagerFactory(entityManagerFactory);
        return txManager;
    }

}
```

Листинг 2 – Конфигурация сервера

```
package ru.nikitin.configs;

import lombok.Data;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.ApplicationContext;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;
import org.springframework.web.servlet.config.annotation.EnableWebMvc;
import
org.springframework.web.servlet.config.annotation.ResourceHandlerRegistry;
import org.springframework.web.servlet.config.annotation.ViewResolverRegistry;
import org.springframework.web.servlet.config.annotation.WebMvcConfigurer;
import org.thymeleaf.spring5.SpringTemplateEngine;
import org.thymeleaf.spring5.templateresolver.SpringResourceTemplateResolver;
import org.thymeleaf.spring5.view.ThymeleafViewResolver;

@Configuration
@EnableWebMvc
@ComponentScan("ru.nikitin.controllers")
public class WebConfig implements WebMvcConfigurer {
    private final ApplicationContext applicationContext;
    @Autowired
    public WebConfig(ApplicationContext applicationContext) {
        this.applicationContext = applicationContext;
    }

    @Bean
    public SpringResourceTemplateResolver templateResolver() {
        SpringResourceTemplateResolver templateResolver = new
                SpringResourceTemplateResolver();
        templateResolver.setApplicationContext(applicationContext);
        templateResolver.setPrefix("/WEB-INF/views/");
        templateResolver.setSuffix(".html");
        return templateResolver;
    }

    @Bean
    public SpringTemplateEngine templateEngine() {
        SpringTemplateEngine templateEngine = new SpringTemplateEngine();
        SpringResourceTemplateResolver resolver = templateResolver();
```

Окончание листинга 2

```
        resolver.setCharacterEncoding("UTF-8");
        templateEngine.setTemplateResolver(resolver);
        templateEngine.setEnableSpringELCompiler(true);
        return templateEngine;
    }
    @Override
    public void configureViewResolvers(ViewResolverRegistry registry) {
        ThymeleafViewResolver resolver = new ThymeleafViewResolver();
        resolver.setTemplateEngine(templateEngine());
        resolver.setCharacterEncoding("UTF-8");
        resolver.setContentType("text/html;charset=UTF-8");
        registry.viewResolver(resolver);
    }
}
```

Листинг 3 – Конфигурация сервлета

```
package ru.nikitin.configs;

import
org.springframework.web.servlet.support.AbstractAnnotationConfigDispatcherServle
tInitializer;

public class DispatcherConfig extends
AbstractAnnotationConfigDispatcherServletInitializer {
    @Override
    protected Class<?>[] getRootConfigClasses() {
        return new Class<?>[] { AppConfig.class };
    }
    @Override
    protected Class<?>[] getServletConfigClasses() {
        return new Class<?>[] { WebConfig.class };
    }
    @Override
    protected String[] getServletMappings() {
        return new String[] { "/" };
    }
}
```

Листинг 4 – Конфигурация безопасности

```
package ru.nikitin.configs;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Qualifier;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.authentication.AuthenticationProvider;
import
org.springframework.security.authentication.dao.DaoAuthenticationProvider;
import
org.springframework.security.config.annotation.authentication.builders.Authentic
ationManagerBuilder;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import
org.springframework.security.config.annotation.web.configuration.EnableWebSecuri
ty;
import
org.springframework.security.config.annotation.web.configuration.WebSecurityConf
igurerAdapter;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.thymeleaf.extras.springsecurity5.dialect.SpringSecurityDialect;

@Configuration
@ComponentScan("ru.nikitin")
@EnableWebSecurity
public class SecurityConfig extends WebSecurityConfigurerAdapter {
    @Qualifier("userServiceImpl")
    @Autowired
    private UserDetailsService userDetailsService;

    @Autowired
    private AuthenticationProvider authProvider;

    @Bean("passwordEncoder")
    public PasswordEncoder passwordEncoder() {
        return new BCryptPasswordEncoder();
    }
```

Продолжение листинга 4

```
    @Override
    protected void configure(AuthenticationManagerBuilder auth) {
        DaoAuthenticationProvider authProvider = new DaoAuthenticationProvider();
        authProvider.setUserDetailsService(userDetailsService);
        authProvider.setPasswordEncoder(passwordEncoder());
    }

    @Override
    protected void configure(final HttpSecurity http) throws Exception {
        http.csrf().disable()
                .authorizeRequests()
                .antMatchers("/register").permitAll()
                .antMatchers("/auth").permitAll()
                .antMatchers("/home", "/show_criteria").hasAnyAuthority("ADMIN",
"USER")
                .antMatchers("/add", "/edit", "/delete").hasAuthority("ADMIN")
                .anyRequest().authenticated()
                .and()
                .formLogin()
                .loginPage("/auth")
                .defaultSuccessUrl("/home")
                .permitAll()
                .and()
                .logout()
                .permitAll()
                .logoutSuccessUrl("/auth")
                .and().authenticationProvider(authProvider);;
    }

    @Bean
    public DaoAuthenticationProvider authenticationProvider() {
        DaoAuthenticationProvider authProvider = new DaoAuthenticationProvider();
        authProvider.setUserDetailsService(userDetailsService);
        authProvider.setPasswordEncoder(passwordEncoder());

        return authProvider;
    }

    @Autowired
    protected void configureGlobal(AuthenticationManagerBuilder auth) throws
Exception {
```

Окончание листинга 4

```
auth.userDetailsService(userDetailsService).passwordEncoder(passwordEncoder());
    }

    @Bean
    public SpringSecurityDialect springSecurityDialect() {
        return new SpringSecurityDialect();
    }
}
```

Листинг 5 – Конфигурация авторизации пользователя

```
package ru.nikitin.configs;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.authentication.AuthenticationProvider;
import org.springframework.security.authentication.BadCredentialsException;
import
org.springframework.security.authentication.UsernamePasswordAuthenticationToken;
import org.springframework.security.core.Authentication;
import org.springframework.security.core.AuthenticationException;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.stereotype.Component;

@Component
public class AuthProviderAdapter implements AuthenticationProvider {

    @Autowired
    private PasswordEncoder encoder;

    @Autowired
    private UserDetailsService service;

    @Override
    public Authentication authenticate(Authentication authentication) throws
AuthenticationException {
        String username = authentication.getName();
        String password = authentication.getCredentials().toString();
        UserDetails u = service.loadUserByUsername(username);
```

Окончание листинга 5

```
        if (!encoder.matches(password, u.getPassword())) {
            throw new BadCredentialsException("Invalid credentials.");
        }

        return new UsernamePasswordAuthenticationToken(username, password,
u.getAuthorities());
    }

    @Override
    public boolean supports(Class<?> authentication) {
        return authentication.equals(UsernamePasswordAuthenticationToken.class);
    }
}
```

На листингах 6-10 представлен код программы, реализующий задание.

Листинг 6 – Класс, реализующий обращения к БД

```
package ru.nikitin.services.impl;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.data.domain.Sort;
import org.springframework.stereotype.Service;
import ru.nikitin.services.StationeryService;
import ru.nikitin.entities.Stationery;
import ru.nikitin.repos.StationeryRepository;

import java.util.List;
import java.util.Optional;

@Service
public class StationeryServiceImpl implements StationeryService {
    @Autowired
    private StationeryRepository stationeryRepository;

    @Override
    public boolean add(Stationery stationery) {
        this.stationeryRepository.save(stationery);
        return true;
    }
```

Окончание листинга 6

```
@Override
    public boolean delete(Integer id) {
        if (checkId(id)) {
            this.stationeryRepository.deleteById(id);
            return true;
        }
        return false;
    }

    @Override
    public boolean update(Integer id, Stationery stationery) {
        if (checkId(id)) {
            this.stationeryRepository.deleteById(id);
            this.stationeryRepository.save(stationery);
            return true;
        }
        return false;
    }

    @Override
    public List<Stationery> getByManufacturer(String manufacturer) {
        return
this.stationeryRepository.findStationeryByManufacturer(manufacturer);
    }
    @Override
    public List<Stationery> getAll() {
        return this.stationeryRepository.findAll(Sort.by("type"));
    }

    public boolean checkId(int id) {
        return this.stationeryRepository.findById(id).isPresent();
    }

    public Optional<Stationery> get(Integer id) {
        return this.stationeryRepository.findById(id);
    }
}
```

Листинг 7 – Класс-сущность канцелярского предмета

```
package ru.nikitin.entities;

import lombok.AccessLevel;
import lombok.Setter;

import javax.persistence.*;
import javax.validation.constraints.Min;

@Entity
@Table(name = "stationery")
public class Stationery {
    @Id
    @Column(name = "id")
    @GeneratedValue(strategy= GenerationType.IDENTITY)
    @Setter(AccessLevel.NONE)
    private Integer id;

    @Column(name = "type")
    private String type;

    @Column(name = "price")
    @Min(0)
    private Double price;

    @Column(name = "amount")
    @Min(0)
    private Integer amount;

    @Column(name = "subtype")
    private String subtype;

    @Column(name = "manufacturer")
    private String manufacturer;

    public Stationery() {
    }

    public Stationery(String type, String subtype, Double price, Integer amount,
String manufacturer) {
        this.type = type;
        this.price = price;
```

Продолжение листинга 7

```
        this.amount = amount;
        this.subtype = subtype;
        this.manufacturer = manufacturer;
    }

    public Integer getId() {
        return id;
    }

    public String getType() {
        return type;
    }

    public Double getPrice() {
        return price;
    }

    public Integer getAmount() {
        return amount;
    }

    public String getSubtype() {
        return subtype;
    }

    public String getManufacturer() {
        return manufacturer;
    }

    public void setType(String type) {
        this.type = type;
    }

    public void setPrice(Double price) {
        this.price = price;
    }

    public void setAmount(Integer amount) {
        this.amount = amount;
    }
```

Окончание листинга 7

```
    public void setSubtype(String subtype) {
        this.subtype = subtype;
    }

    public void setManufacturer(String manufacturer) {
        this.manufacturer = manufacturer;
    }

    @Override
    public String toString() {
        return "id: " + this.id + " | " + "type: " + this.type + " | " + "subtype:
" + this.subtype + " | "
                + "price: " + this.price + " | " + "manufacturer: " +
this.manufacturer
                + " | " + "amount: " + this.amount;
    }
}
```

Листинг 8 – Класс-репозиторий БД

```
package ru.nikitin.repos;

import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;
import ru.nikitin.entities.Stationery;

import java.util.List;

@Repository
public interface StationeryRepository extends JpaRepository<Stationery, Integer>
{
    List<Stationery> findStationeryByManufacturer(String manufacturer);
}
```

Листинг 9 – Класс, реализующий обращения к БД

```
package ru.nikitin.services.impl;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.data.domain.Sort;
import org.springframework.stereotype.Service;
import ru.nikitin.services.StationeryService;
import ru.nikitin.entities.Stationery;
```

Продолжение листинга 9

```
import ru.nikitin.repos.StationeryRepository;

import java.util.List;
import java.util.Optional;

@Service
public class StationeryServiceImpl implements StationeryService {
    @Autowired
    private StationeryRepository stationeryRepository;

    @Override
    public boolean add(Stationery stationery) {
        this.stationeryRepository.save(stationery);
        return true;
    }

    @Override
    public boolean delete(Integer id) {
        if (checkId(id)) {
            this.stationeryRepository.deleteById(id);
            return true;
        }
        return false;
    }

    @Override
    public boolean update(Integer id, Stationery stationery) {
        if (checkId(id)) {
            this.stationeryRepository.deleteById(id);
            this.stationeryRepository.save(stationery);
            return true;
        }
        return false;
    }

    @Override
    public List<Stationery> getByManufacturer(String manufacturer) {
        return
this.stationeryRepository.findStationeryByManufacturer(manufacturer);
    }
    @Override
```

Окончание листинга 9

```
    public List<Stationery> getAll() {
        return this.stationeryRepository.findAll(Sort.by("type"));
    }

    public boolean checkId(int id) {
        return this.stationeryRepository.findById(id).isPresent();
    }

    public Optional<Stationery> get(Integer id) {
        return this.stationeryRepository.findById(id);
    }
}
```

Листинг 10 – Класс-контроллер, реализующий серверную логику

```
package ru.nikitin.controllers;

import lombok.Data;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Qualifier;
import org.springframework.context.ApplicationContext;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.ui.ModelMap;
import org.springframework.validation.BindingResult;
import org.springframework.web.bind.annotation.*;
import ru.nikitin.entities.Stationery;
import ru.nikitin.services.UserService;
import ru.nikitin.services.impl.StationeryServiceImpl;

import javax.validation.Valid;

@Controller
public @Data
class MainController {
    private final PasswordEncoder passwordEncoder;
    private final ApplicationContext context;
    private final StationeryServiceImpl stationeryService;
    private final UserService userService;

    @Autowired
```

Продолжение листинга 10

```
    public MainController(@Qualifier("passwordEncoder") PasswordEncoder
passwordEncoder,
                          @Qualifier("userServiceImpl") UserService userService,
                          ApplicationContext applicationContext) {
        this.passwordEncoder = passwordEncoder;
        this.context = applicationContext;
        this.userService = userService;
        this.stationeryService = context.getBean("stationeryServiceImpl",
StationeryServiceImpl.class);
    }

    @GetMapping("/home")
    public String mainPage(Model model) {
        model.addAttribute("stationery", stationeryService.getAll());
        return "home";
    }

    @GetMapping("/add")
    public String addPage(Model model) {
        model.addAttribute("stationery", new Stationery());
        return "add";
    }
    @PostMapping("/add")
    public String addFormHandler(@ModelAttribute("stationery") Stationery
stationery) {
        stationeryService.add(stationery);
        return "redirect:/home";
    }

    @GetMapping("/edit")
    public String editPage(Model model) {
        model.addAttribute("stationery", new Stationery());
        return "edit";
    }
    @PostMapping("/edit")
    public String editFormHandler(
            @ModelAttribute @Valid Stationery stationery,
            BindingResult bindingResult,
            @RequestParam("id") Integer id) {
        System.out.println(bindingResult);
        if (bindingResult.hasErrors())
```

Продолжение листинга 10

```
            return "redirect:/edit";

        stationeryService.update(id, stationery);
        return "redirect:/home";
    }

    @GetMapping("/delete")
    public String deletePage() {
        return "delete";
    }
    @PostMapping("/delete")
    public String deleteFormHandler( @RequestParam("id") Integer id ) {
        stationeryService.delete(id);
        return "redirect:/home";
    }

    @GetMapping("/show_criteria")
    public String criteriaPage() {
        return "criteria";
    }
    @PostMapping("/show_criteria")
    public String criteriaFormHandler( @RequestParam String manufacturer,
ModelMap model ) {
        model.addAttribute("stationery",
stationeryService.getByManufacturer(manufacturer));
        return "criteria_result";
    }

    @GetMapping("/register")
    public String register() {
        return "registration";
    }

    @PostMapping("/register")
    public String registerHandler(@RequestParam String username, @RequestParam
String password) {
        try {
            boolean registerSuccess = userService.register(username,
passwordEncoder.encode(password));
            if (registerSuccess)
                System.out.println("Создан пользователь " + username);
```

Окончание листинга 10

```
            else {
                System.out.println("Существует пользователь " + username);
                return "registration";
            }
        } catch (Exception e) {
            System.out.println(e.getMessage());
        }
        return "redirect:/auth";
    }

    @GetMapping("/auth")
    public String auth() {
        return "auth";
    }
}
```

Листинг 11 – Класс, реализующий авторизацию и регистрацию

```
package ru.nikitin.services.impl;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.core.userdetails.UsernameNotFoundException;
import org.springframework.stereotype.Service;
import ru.nikitin.entities.UserEntity;
import ru.nikitin.repos.UserRepository;
import ru.nikitin.services.UserService;

@Service("userServiceImpl")
public class UserServiceImpl implements UserDetailsService, UserService {
    @Autowired
    private UserRepository userRepository;

    @Override
    public UserDetails loadUserByUsername(String username)
            throws UsernameNotFoundException {
        UserEntity                          userEntity                          =
userRepository.findByUsername(username).orElse(null);

        if (userEntity == null) {
            throw new UsernameNotFoundException("Пользователь не найден!");
```

Окончание листинга 11

```
        }

        return new UserDetailsImpl(userEntity);
    }

    @Override
    public boolean register(String username, String password) {
        if (userRepository.findByUsername(username).isPresent())
            return false;
        UserEntity userEntity = new UserEntity(username, password);
        userRepository.save(userEntity);
        return true;
    }
}
```

5 Вывод

В результате работы были закреплены на практике знания сетевых запросов, устройства систем регистрации и авторизации и их применения в фреймворке Spring, а также реализована программа по заданию.