МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«МИРЭА – Российский технологический университет»

РТУ МИРЭА

Институт информационных технологий (ИТ)

Кафедра инструментального и прикладного программного обеспечения (ИиППО)

ОТЧЕТ ПО ПРАКТИЧЕСКИМ РАБОТАМ №23 - 24

по дисциплине

«Шаблоны программных платформ языка Java

Вариант 21

Выполнил студент группы ИКБО-20-19 Николаев-Аксенов И. С.

Принял Батанов А. О.
Ассистент

Практические работы выполнены «___»_____2021 г. (подпись студента)

«Зачтено» «___»_____2021 г. (подпись руководителя)

Москва 2021

Содержание

Практическая работа №23

Цель работы

Тема: Использование Spring Security для аутентификации и авторизации пользователей.

Постановка задачи: В приложении из предыдущего задания добавить возможность регистрации и авторизации пользователей, хранение cookie сессий в базе данных PostgreSQL, хеширование паролей алгоритмом Bcrypt, защиту всех запросов, кроме запросов на авторизацию и регистрацию, от неавторизированных пользователей.

Листинг программы

User.java

```java
package app.Application.Classes;

import org.springframework.security.core.GrantedAuthority;
import org.springframework.security.core.userdetails.UserDetails;

import javax.persistence.*;
import javax.validation.constraints.Size;
import java.util.Collection;
import java.util.Set;

@Entity
@Table(name = "t_user")
public class User implements UserDetails {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Size(min=2, message = "Не меньше 5 знаков")
    private String username;

    @Size(min=2, message = "Не меньше 5 знаков")
    private String password;

    @Transient
    private String passwordConfirm;

    @ManyToMany(fetch = FetchType.EAGER)
    private Set<Role> roles;

    public User() {
    }

    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }

    @Override
```

```
    public String getUsername() {
        return username;
    }

    @Override
    public boolean isAccountNonExpired() {
        return true;
    }

    @Override
    public boolean isAccountNonLocked() {
        return true;
    }

    @Override
    public boolean isCredentialsNonExpired() {
        return true;
    }

    @Override
    public boolean isEnabled() {
        return true;
    }

    public void setUsername(String username) {
        this.username = username;
    }

    @Override
    public Collection<? extends GrantedAuthority> getAuthorities() {
        return getRoles();
    }

    @Override
    public String getPassword() {
        return password;
    }

    public void setPassword(String password) {
        this.password = password;
    }

    public String getPasswordConfirm() {
        return passwordConfirm;
    }

    public void setPasswordConfirm(String passwordConfirm) {
        this.passwordConfirm = passwordConfirm;
    }

    public Set<Role> getRoles() {
        return roles;
    }

    public void setRoles(Set<Role> roles) {
        this.roles = roles;
    }

}
```

Role.java

```java
package app.Application.Classes;

import org.springframework.security.core.GrantedAuthority;

import javax.persistence.*;
import java.util.Set;

@Entity
@Table(name = "t_role")
public class Role implements GrantedAuthority {
    @Id
    private Long id;

    private String name;

    @Transient
    @ManyToMany(mappedBy = "roles")
    private Set<User> users;

    public Role() {
    }

    public Role(Long id) {
        this.id = id;
    }

    public Role(Long id, String name) {
        this.id = id;
        this.name = name;
    }

    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public Set<User> getUsers() {
        return users;
    }

    public void setUsers(Set<User> users) {
        this.users = users;
    }

    @Override
    public String getAuthority() {
        return getName();
    }
}
```

UserRepository.java

```java
package app.Application.Interfaces;

import app.Application.Classes.User;
import org.springframework.data.jpa.repository.JpaRepository;

public interface UserRepository extends JpaRepository<User, Long> {
    User findByUsername(String username);
}
```

RoleRepository.java

```java
package app.Application.Interfaces;

import app.Application.Classes.Role;
import org.springframework.data.jpa.repository.JpaRepository;

public interface RoleRepository extends JpaRepository<Role, Long> {
}
```

UserService.java

```java
package app.Application.Services;

import app.Application.Classes.Role;
import app.Application.Classes.User;
import app.Application.Interfaces.RoleRepository;
import app.Application.Interfaces.UserRepository;
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.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.stereotype.Service;

import javax.persistence.EntityManager;
import javax.persistence.PersistenceContext;
import java.util.Collections;
import java.util.List;
import java.util.Optional;

@Service
public class UserService implements UserDetailsService {
    @PersistenceContext
    private EntityManager em;

    @Autowired
    UserRepository userRepository;

    @Autowired
    RoleRepository roleRepository;

    @Autowired
    BCryptPasswordEncoder bCryptPasswordEncoder;

    public UserService(UserRepository userRepository) {
        this.userRepository = userRepository;
    }
```

```
    @Override
    public UserDetails loadUserByUsername(String username) throws
UsernameNotFoundException {
        User user = userRepository.findByUsername(username);

        if (user == null) {
            throw new UsernameNotFoundException("User not found");
        }

        return user;
    }

    public User findUserById(Long userId) {
        Optional<User> userFromDb = userRepository.findById(userId);
        return userFromDb.orElse(new User());
    }

    public List<User> allUsers() {
        return userRepository.findAll();
    }

    public boolean saveUser(User user) {
        User userFromDB = userRepository.findByUsername(user.getUsername());

        if (userFromDB != null) {
            return false;
        }

        user.setRoles(Collections.singleton(new Role(1L, "ROLE_USER")));
        user.setPassword(bCryptPasswordEncoder.encode(user.getPassword()));
        userRepository.save(user);
        return true;
    }

    public boolean deleteUser(Long userId) {
        if (userRepository.findById(userId).isPresent()) {
            userRepository.deleteById(userId);
            return true;
        }
        return false;
    }

    public List<User> usergtList(Long idMin) {
        return em.createQuery("SELECT u FROM User u WHERE u.id > :paramId", User.class)
                .setParameter("paramId", idMin).getResultList();
    }
}
```

AdminController.java

```
package app.Application.Controllers;

import app.Application.Services.UserService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestParam;

@Controller
public class AdminController {
    @Autowired
    private UserService userService;

    @GetMapping("/admin")
    public String userList(Model model) {
        model.addAttribute("allUsers", userService.allUsers());
        return "admin";
    }

    @PostMapping("/admin")
    public String  deleteUser(@RequestParam(required = true, defaultValue = "" ) Long
userId,
                              @RequestParam(required = true, defaultValue = "" ) String
action,
                              Model model) {
        if (action.equals("delete")){
            userService.deleteUser(userId);
        }
        return "redirect:/admin";
    }

    @GetMapping("/admin/gt/{userId}")
    public String  gtUser(@PathVariable("userId") Long userId, Model model) {
        model.addAttribute("allUsers", userService.usergtList(userId));
        return "admin";
    }
}
```

CookieController.java

```
package app.Application.Controllers;

import org.springframework.web.bind.annotation.RestController;

@RestController
public class CookieController {
}
```

RegistrationController.java

```
package app.Application.Controllers;

import app.Application.Classes.User;
import app.Application.Services.UserService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.validation.BindingResult;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.PostMapping;

import javax.validation.Valid;

@Controller
public class RegistrationController {

    @Autowired
    private UserService userService;

    @GetMapping("/registration")
    public String registration(Model model) {
        model.addAttribute("userForm", new User());

        return "registration";
    }

    @PostMapping("/registration")
    public String addUser(@ModelAttribute("userForm") @Valid User userForm,
BindingResult bindingResult, Model model) {

        if (bindingResult.hasErrors()) {
            return "registration";
        }
        if (!userForm.getPassword().equals(userForm.getPasswordConfirm())){
            model.addAttribute("passwordError", "Пароли не совпадают");
            return "registration";
        }
        if (!userService.saveUser(userForm)){
            model.addAttribute("usernameError", "Пользователь с таким именем уже
существует");
            return "registration";
        }

        return "redirect:/";
    }
}
```

Config.java

```
package app.Application.Configuration;

import org.springframework.context.annotation.Configuration;
import org.springframework.context.annotation.EnableAspectJAutoProxy;
import org.springframework.data.jpa.repository.config.EnableJpaRepositories;
import org.springframework.scheduling.annotation.EnableAsync;
import
org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAd
apter;

@Configuration
@EnableAspectJAutoProxy
@EnableJpaRepositories(basePackages = {"app.Application"})
@EnableAsync
public class Config extends WebSecurityConfigurerAdapter {
}
```

MvcConfig.java

```
package app.Application.Configuration;

import org.springframework.context.annotation.Configuration;
import org.springframework.web.servlet.config.annotation.ViewControllerRegistry;
import org.springframework.web.servlet.config.annotation.WebMvcConfigurer;

@Configuration
public class MvcConfig implements WebMvcConfigurer {

    @Override
    public void addViewControllers(ViewControllerRegistry registry) {
        registry.addViewController("/login").setViewName("login");
        registry.addViewController("/news").setViewName("news");
    }
}
```

WebSecurityConfig.java

```
package app.Application.Configuration;

import app.Application.Services.UserService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.core.annotation.Order;
import org.springframework.security.authentication.AuthenticationManager;
import
org.springframework.security.config.annotation.authentication.builders.AuthenticationMan
agerBuilder;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import
org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
import
org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAd
apter;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;

@Configuration
```

```java
@EnableWebSecurity
@Order(1000)
public class WebSecurityConfig extends WebSecurityConfigurerAdapter {
    @Autowired
    UserService userService;
    @Bean("authenticationManager")
    @Override
    public AuthenticationManager authenticationManagerBean() throws Exception {
        return super.authenticationManagerBean();
    }
    @Bean
    public BCryptPasswordEncoder bCryptPasswordEncoder() {
        return new BCryptPasswordEncoder();
    }

    @Override
    protected void configure(HttpSecurity httpSecurity) throws Exception {
        httpSecurity
                .csrf().disable().cors().disable()
                .authorizeRequests()
                .antMatchers("/registration").permitAll()
                .anyRequest().authenticated()
                .and()
                .formLogin()
                .loginPage("/login")
                .defaultSuccessUrl("/")
                .permitAll()
                .and()
                .logout().deleteCookies("JSESSIONID")
                .permitAll()
                .logoutSuccessUrl("/")
                .and()
                .rememberMe().key("uniqueAndSecret");
    }

    @Autowired
    protected void configureGlobal(AuthenticationManagerBuilder auth) throws Exception {
        auth.userDetailsService(userService).passwordEncoder(bCryptPasswordEncoder());
    }
}
```

Aspect.java

```java
package app.Application;

import lombok.extern.slf4j.Slf4j;
import org.aspectj.lang.ProceedingJoinPoint;
import org.aspectj.lang.annotation.Around;
import org.aspectj.lang.annotation.Pointcut;
import org.springframework.stereotype.Component;

import java.util.logging.Logger;

@Slf4j
@Component
@org.aspectj.lang.annotation.Aspect
public class Aspect {
    private Logger log = Logger.getLogger(Aspect.class.getName());

    @Around("allServiceMethods()")
    public Object logExecutionTime (ProceedingJoinPoint joinPoint) throws Throwable {
        long start = System.currentTimeMillis();
        Object proceed = joinPoint.proceed();
        long executionTime = System.currentTimeMillis() - start;
        //log.log(Level.INFO, joinPoint.getSignature() + " выполнен за " + executionTime
+ "мс");
        log.info(joinPoint.getSignature() + " выполнен за " + executionTime + "мс");
        return proceed;
    }

    @Pointcut("within(Homework.twentieth.Services.*)")
    public void allServiceMethods() {}
}
```

Результат выполнения программы

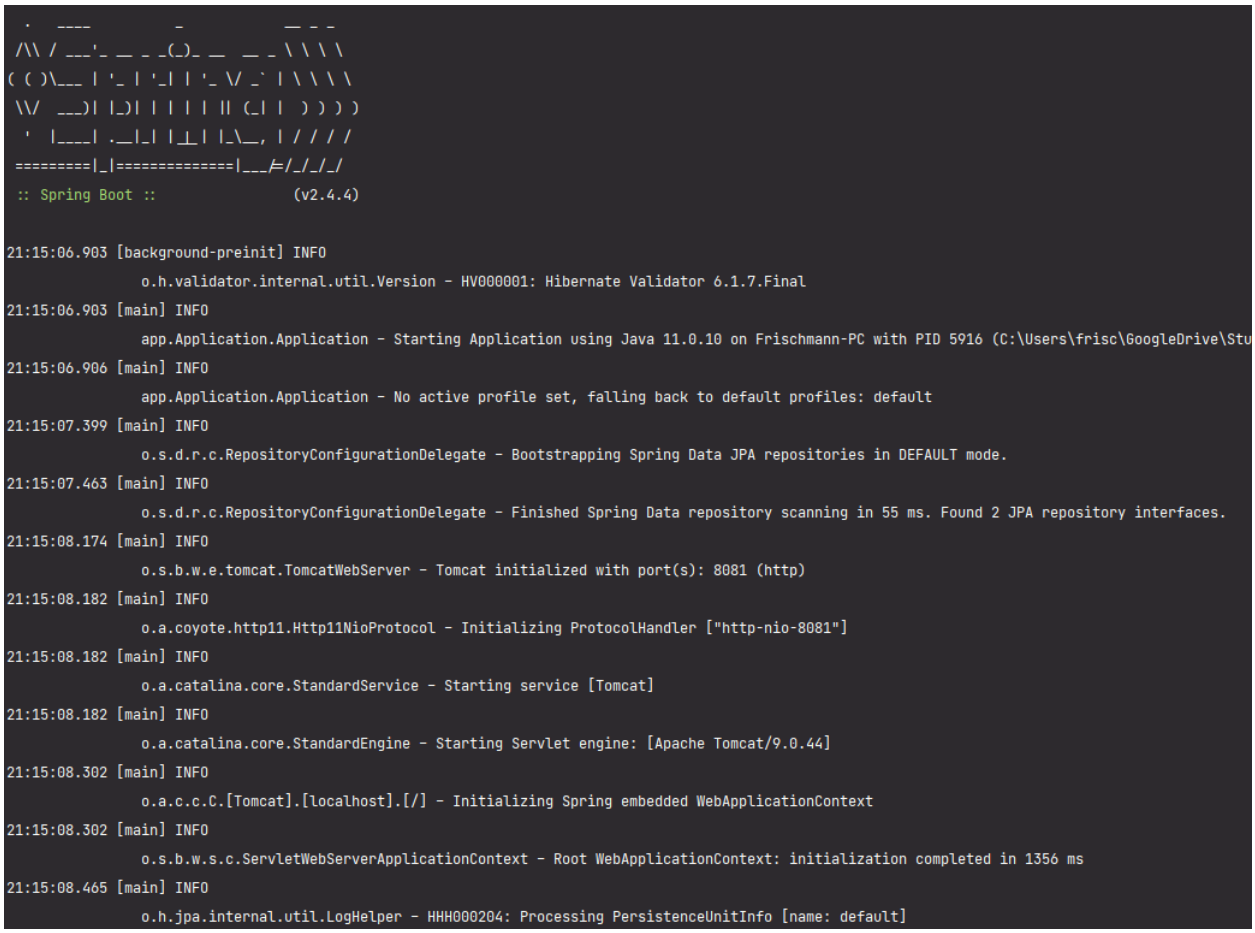

Рисунок 23.1 – Демонстрация работы программы

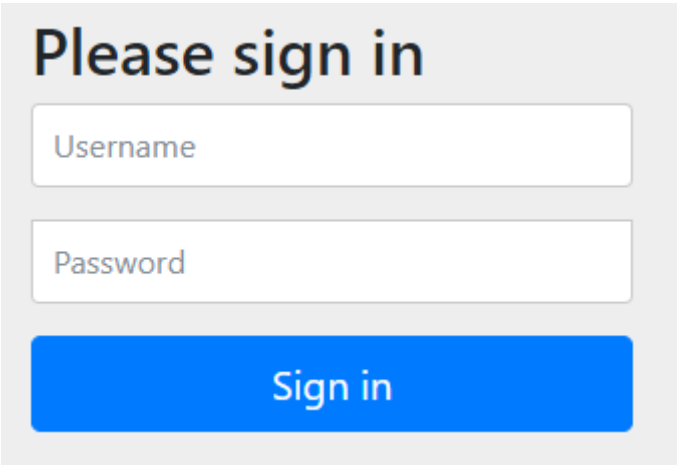

Рисунок 23.2 – Демонстрация работы программы

Практическая работа №24

Цель работы

Тема: Тестирование в Spring Framework с использованием Junit.

Постановка задачи: Написать модульное тестирование для всех классов сервисов приложения из предыдущего задания.

Листинг программы

User.java

```
package app.Application.Classes;

import org.springframework.security.core.GrantedAuthority;
import org.springframework.security.core.userdetails.UserDetails;

import javax.persistence.*;
import javax.validation.constraints.Size;
import java.util.Collection;
import java.util.Set;

@Entity
@Table(name = "t_user")
public class User implements UserDetails {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Size(min=2, message = "Не меньше 5 знаков")
    private String username;

    @Size(min=2, message = "Не меньше 5 знаков")
    private String password;

    @Transient
    private String passwordConfirm;

    @ManyToMany(fetch = FetchType.EAGER)
    private Set<Role> roles;

    public User() {
    }

    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }

    @Override
    public String getUsername() {
        return username;
    }
```

```
    @Override
    public boolean isAccountNonExpired() {
        return true;
    }

    @Override
    public boolean isAccountNonLocked() {
        return true;
    }

    @Override
    public boolean isCredentialsNonExpired() {
        return true;
    }

    @Override
    public boolean isEnabled() {
        return true;
    }

    public void setUsername(String username) {
        this.username = username;
    }

    @Override
    public Collection<? extends GrantedAuthority> getAuthorities() {
        return getRoles();
    }

    @Override
    public String getPassword() {
        return password;
    }

    public void setPassword(String password) {
        this.password = password;
    }

    public String getPasswordConfirm() {
        return passwordConfirm;
    }

    public void setPasswordConfirm(String passwordConfirm) {
        this.passwordConfirm = passwordConfirm;
    }

    public Set<Role> getRoles() {
        return roles;
    }

    public void setRoles(Set<Role> roles) {
        this.roles = roles;
    }

}
```

Role.java

```java
package app.Application.Classes;

import org.springframework.security.core.GrantedAuthority;

import javax.persistence.*;
import java.util.Set;

@Entity
@Table(name = "t_role")
public class Role implements GrantedAuthority {
    @Id
    private Long id;

    private String name;

    @Transient
    @ManyToMany(mappedBy = "roles")
    private Set<User> users;

    public Role() {
    }

    public Role(Long id) {
        this.id = id;
    }

    public Role(Long id, String name) {
        this.id = id;
        this.name = name;
    }

    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public Set<User> getUsers() {
        return users;
    }

    public void setUsers(Set<User> users) {
        this.users = users;
    }

    @Override
    public String getAuthority() {
        return getName();
    }
}
```

UserRepository.java

```java
package app.Application.Interfaces;

import app.Application.Classes.User;
import org.springframework.data.jpa.repository.JpaRepository;

public interface UserRepository extends JpaRepository<User, Long> {
    User findByUsername(String username);
}
```

RoleRepository.java

```java
package app.Application.Interfaces;

import app.Application.Classes.Role;
import org.springframework.data.jpa.repository.JpaRepository;

public interface RoleRepository extends JpaRepository<Role, Long> {
}
```

UserService.java

```java
package app.Application.Services;

import app.Application.Classes.Role;
import app.Application.Classes.User;
import app.Application.Interfaces.RoleRepository;
import app.Application.Interfaces.UserRepository;
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.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.stereotype.Service;

import javax.persistence.EntityManager;
import javax.persistence.PersistenceContext;
import java.util.Collections;
import java.util.List;
import java.util.Optional;

@Service
public class UserService implements UserDetailsService {
    @PersistenceContext
    private EntityManager em;

    @Autowired
    UserRepository userRepository;

    @Autowired
    RoleRepository roleRepository;

    @Autowired
    BCryptPasswordEncoder bCryptPasswordEncoder;

    public UserService(UserRepository userRepository) {
        this.userRepository = userRepository;
    }
```

```
    @Override
    public UserDetails loadUserByUsername(String username) throws
UsernameNotFoundException {
        User user = userRepository.findByUsername(username);

        if (user == null) {
            throw new UsernameNotFoundException("User not found");
        }

        return user;
    }

    public User findUserById(Long userId) {
        Optional<User> userFromDb = userRepository.findById(userId);
        return userFromDb.orElse(new User());
    }

    public List<User> allUsers() {
        return userRepository.findAll();
    }

    public boolean saveUser(User user) {
        User userFromDB = userRepository.findByUsername(user.getUsername());

        if (userFromDB != null) {
            return false;
        }

        user.setRoles(Collections.singleton(new Role(1L, "ROLE_USER")));
        user.setPassword(bCryptPasswordEncoder.encode(user.getPassword()));
        userRepository.save(user);
        return true;
    }

    public boolean deleteUser(Long userId) {
        if (userRepository.findById(userId).isPresent()) {
            userRepository.deleteById(userId);
            return true;
        }
        return false;
    }

    public List<User> usergtList(Long idMin) {
        return em.createQuery("SELECT u FROM User u WHERE u.id > :paramId", User.class)
                .setParameter("paramId", idMin).getResultList();
    }
}
```

AdminController.java

```
package app.Application.Controllers;

import app.Application.Services.UserService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestParam;

@Controller
public class AdminController {
    @Autowired
    private UserService userService;

    @GetMapping("/admin")
    public String userList(Model model) {
        model.addAttribute("allUsers", userService.allUsers());
        return "admin";
    }

    @PostMapping("/admin")
    public String  deleteUser(@RequestParam(required = true, defaultValue = "" ) Long
userId,
                              @RequestParam(required = true, defaultValue = "" ) String
action,
                              Model model) {
        if (action.equals("delete")){
            userService.deleteUser(userId);
        }
        return "redirect:/admin";
    }

    @GetMapping("/admin/gt/{userId}")
    public String  gtUser(@PathVariable("userId") Long userId, Model model) {
        model.addAttribute("allUsers", userService.usergtList(userId));
        return "admin";
    }
}
```

CookieController.java

```
package app.Application.Controllers;

import org.springframework.web.bind.annotation.RestController;

@RestController
public class CookieController {
}
```

RegistrationController.java

```
package app.Application.Controllers;

import app.Application.Classes.User;
import app.Application.Services.UserService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.validation.BindingResult;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.PostMapping;

import javax.validation.Valid;

@Controller
public class RegistrationController {

    @Autowired
    private UserService userService;

    @GetMapping("/registration")
    public String registration(Model model) {
        model.addAttribute("userForm", new User());

        return "registration";
    }

    @PostMapping("/registration")
    public String addUser(@ModelAttribute("userForm") @Valid User userForm,
BindingResult bindingResult, Model model) {

        if (bindingResult.hasErrors()) {
            return "registration";
        }
        if (!userForm.getPassword().equals(userForm.getPasswordConfirm())){
            model.addAttribute("passwordError", "Пароли не совпадают");
            return "registration";
        }
        if (!userService.saveUser(userForm)){
            model.addAttribute("usernameError", "Пользователь с таким именем уже
существует");
            return "registration";
        }

        return "redirect:/";
    }
}
```

Config.java

```
package app.Application.Configuration;

import org.springframework.context.annotation.Configuration;
import org.springframework.context.annotation.EnableAspectJAutoProxy;
import org.springframework.data.jpa.repository.config.EnableJpaRepositories;
import org.springframework.scheduling.annotation.EnableAsync;
import
org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAd
apter;

@Configuration
@EnableAspectJAutoProxy
@EnableJpaRepositories(basePackages = {"app.Application"})
@EnableAsync
public class Config extends WebSecurityConfigurerAdapter {
}
```

MvcConfig.java

```
package app.Application.Configuration;

import org.springframework.context.annotation.Configuration;
import org.springframework.web.servlet.config.annotation.ViewControllerRegistry;
import org.springframework.web.servlet.config.annotation.WebMvcConfigurer;

@Configuration
public class MvcConfig implements WebMvcConfigurer {

    @Override
    public void addViewControllers(ViewControllerRegistry registry) {
        registry.addViewController("/login").setViewName("login");
        registry.addViewController("/news").setViewName("news");
    }
}
```

WebSecurityConfig.java

```
package app.Application.Configuration;

import app.Application.Services.UserService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.core.annotation.Order;
import org.springframework.security.authentication.AuthenticationManager;
import
org.springframework.security.config.annotation.authentication.builders.AuthenticationMan
agerBuilder;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import
org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
import
org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAd
apter;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;

@Configuration
```

```java
@EnableWebSecurity
@Order(1000)
public class WebSecurityConfig extends WebSecurityConfigurerAdapter {
    @Autowired
    UserService userService;
    @Bean("authenticationManager")
    @Override
    public AuthenticationManager authenticationManagerBean() throws Exception {
        return super.authenticationManagerBean();
    }
    @Bean
    public BCryptPasswordEncoder bCryptPasswordEncoder() {
        return new BCryptPasswordEncoder();
    }

    @Override
    protected void configure(HttpSecurity httpSecurity) throws Exception {
        httpSecurity
                .csrf().disable().cors().disable()
                .authorizeRequests()
                .antMatchers("/registration").permitAll()
                .anyRequest().authenticated()
                .and()
                .formLogin()
                .loginPage("/login")
                .defaultSuccessUrl("/")
                .permitAll()
                .and()
                .logout().deleteCookies("JSESSIONID")
                .permitAll()
                .logoutSuccessUrl("/")
                .and()
                .rememberMe().key("uniqueAndSecret");
    }

    @Autowired
    protected void configureGlobal(AuthenticationManagerBuilder auth) throws Exception {
        auth.userDetailsService(userService).passwordEncoder(bCryptPasswordEncoder());
    }
}
```

Aspect.java

```java
package app.Application;

import lombok.extern.slf4j.Slf4j;
import org.aspectj.lang.ProceedingJoinPoint;
import org.aspectj.lang.annotation.Around;
import org.aspectj.lang.annotation.Pointcut;
import org.springframework.stereotype.Component;

import java.util.logging.Logger;

@Slf4j
@Component
@org.aspectj.lang.annotation.Aspect
public class Aspect {
    private Logger log = Logger.getLogger(Aspect.class.getName());

    @Around("allServiceMethods()")
    public Object logExecutionTime (ProceedingJoinPoint joinPoint) throws Throwable {
        long start = System.currentTimeMillis();
        Object proceed = joinPoint.proceed();
        long executionTime = System.currentTimeMillis() - start;
        //log.log(Level.INFO, joinPoint.getSignature() + " выполнен за " + executionTime
+ "мс");
        log.info(joinPoint.getSignature() + " выполнен за " + executionTime + "мс");
        return proceed;
    }

    @Pointcut("within(Homework.twentieth.Services.*)")
    public void allServiceMethods() {}
}
```

UserServiceImplTest.java

```java
package app.Application.Test;

import app.Application.Classes.User;
import app.Application.Interfaces.UserRepository;
import app.Application.Services.UserService;
import org.junit.jupiter.api.Assertions;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.extension.ExtendWith;
import org.mockito.Mock;
import org.mockito.Mockito;
import org.mockito.junit.jupiter.MockitoExtension;

import java.util.List;

import static org.mockito.Mockito.mock;

@ExtendWith(MockitoExtension.class)
public class UserServiceImplTest {
    @Mock
    private UserRepository userRepository;

    @Test
    public void getGame() {
        userRepository=mock(UserRepository.class);

        User user = new User();
        user.setUsername("Ivan");

        User user1 = new User();
        user1.setUsername("Petr");

        Mockito.when(userRepository.findAll()).thenReturn(List.of(user,user1));
        UserService userService = new UserService(userRepository);
        Assertions.assertEquals(2, userService.allUsers().size());
        Assertions.assertEquals("Petr",
                userService.allUsers().get(0).getUsername());
    }
}
```

Результат выполнения программы

```
> Task :compileJava UP-TO-DATE
> Task :processResources UP-TO-DATE
> Task :classes UP-TO-DATE
> Task :compileTestJava NO-SOURCE
> Task :processTestResources NO-SOURCE
> Task :testClasses UP-TO-DATE
> Task :test NO-SOURCE
BUILD SUCCESSFUL in 1s
2 actionable tasks: 2 up-to-date
21:42:03: Task execution finished 'test'.
```

Рисунок 24.1 – Демонстрация работы программы

Вывод

В ходе выполнения данных практических работ были получены навыки работы с основными технологиями, необходимыми для создания клиент-серверных приложений. Также были получены навыки работы с фреймворком Spring.

Список использованных источников

1. Стелтинг С., Маасен О. Применение шаблонов Java. Библиотека профессионала.: Пер. с англ. — М.: Издательский дом "Вильяме", 2002. — 576 с.: ил. — Парал. тит. англ.
2. Functional Interfaces in Java: Fundamentals and Examples 1st ed. Edition, Kindle Edition [Электронный ресурс]. URL: https://www.amazon.com/Functional-Interfaces-Java-Fundamentals-Examples-ebook/dp/B07NRHQSCW (дата обращения: 29.01.21). Заголовок с экрана.
3. Hibernate Search 6.0.0.Final: Reference Documentation [Электронный ресурс]. URL: https://docs.jboss.org/hibernate/stable/search/reference/en-US/html_single/ (дата обращения: 29.01.21). Заголовок с экрана.
4. Паттерны проектирования на Java. Каталог Java-примеров. [Электронный ресурс]. URL: https://refactoring.guru/ru/design-patterns/java (дата обращения: 29.01.21). Заголовок с экрана.
5. Руководство по Spring [Электронный ресурс]. URL: https://proselyte.net/tutorials/spring-tutorial-full-version/ (дата обращения: 29.01.21). Заголовок с экрана.
6. The Reactive Manifesto [Электронный ресурс]. URL: https://www.reactivemanifesto.org/ (дата обращения: 29.01.21). Заголовок с экрана.
7. Spring Framework Documentation [Электронный ресурс]. URL: https://docs.spring.io/spring-framework/docs/current/reference/html/web.html (дата обращения: 29.01.21). Заголовок с экрана.
8. Hibernate Search 6.0.0. Final: Reference Documentation [Электронный ресурс]. URL: https://docs.jboss.org/hibernate/stable/search/reference/en-US/html_single/ (дата обращения: 29.01.21). Заголовок с экрана.