МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное бюджетное образовательное учреждение

высшего профессионального образования

**«УЛЬЯНОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»**

Факультет информационных систем и технологий

Кафедра «Информационные системы»

Дисциплина «Интернет программирование»

**Отчёт по лабораторной работе №5**

**Аутентификация и авторизация**

Выполнил студент

гр. ПИбд-21:

Сергеев Н.И.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Научный руководитель:

Филиппов А.А.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Ульяновск

2022

**Задание**

1. Добавить в приложения (SPA и MVC) механизмы аутентификации и авторизации через Spring Security. В MVC-приложении необходимо использовать базовую аутентификацию на основе формы, а в SPA-приложении аутентификацию на основе JWT-токена.
2. Для аутентификации следует разработать отдельную страницу и контроллер. Для создания регистрации пользователя можно создать страницу и контроллер или создавать пользователей с различными ролями при старте приложения (автоматически). При создании пользователя требуется указывать роль для реализации механизмов авторизации.

**Сервер (MVC)**

PasswordEncoderConfiguration.java

package ip.labwork.configuration;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;

import org.springframework.security.crypto.password.PasswordEncoder;

@Configuration

public class PasswordEncoderConfiguration {

@Bean

public PasswordEncoder createPasswordEncoder() {

return new BCryptPasswordEncoder();

}

}

SecurityConfiguration.java

package ip.labwork.configuration;

import ip.labwork.user.controller.UserSignupMvcController;

import ip.labwork.user.model.UserRole;

import ip.labwork.user.service.UserService;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

import org.springframework.http.HttpMethod;

import org.springframework.security.authentication.AuthenticationManager;

import org.springframework.security.config.annotation.authentication.builders.AuthenticationManagerBuilder;

import org.springframework.security.config.annotation.method.configuration.EnableGlobalMethodSecurity;

import org.springframework.security.config.annotation.web.builders.HttpSecurity;

import org.springframework.security.config.annotation.web.builders.WebSecurity;

import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;

import org.springframework.security.config.annotation.web.configuration.WebSecurityCustomizer;

import org.springframework.security.core.userdetails.User;

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

import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;

import org.springframework.security.provisioning.InMemoryUserDetailsManager;

import org.springframework.security.web.SecurityFilterChain;

@Configuration

@EnableWebSecurity

@EnableGlobalMethodSecurity(securedEnabled = true)

public class SecurityConfiguration {

private final Logger log = LoggerFactory.getLogger(SecurityConfiguration.class);

private static final String LOGIN\_URL = "/login";

private final UserService userService;

public SecurityConfiguration(UserService userService) {

this.userService = userService;

createAdminOnStartup();

}

private void createAdminOnStartup() {

final String admin = "admin";

if (userService.findByLogin(admin) == null) {

log.info("Admin user successfully created");

userService.createUser(admin, admin, admin, UserRole.ADMIN);

}

}

@Bean

public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {

http.headers().frameOptions().sameOrigin().and()

.cors().and()

.csrf().disable()

.authorizeRequests()

.requestMatchers(UserSignupMvcController.SIGNUP\_URL).permitAll()

.requestMatchers(HttpMethod.GET, LOGIN\_URL).permitAll()

.anyRequest().authenticated()

.and()

.formLogin()

.loginPage(LOGIN\_URL).permitAll()

.and()

.logout().permitAll();

return http.build();

}

@Bean

public AuthenticationManager authenticationManager(HttpSecurity http, PasswordEncoderConfiguration bCryptPasswordEncoder)

throws Exception {

return http.getSharedObject(AuthenticationManagerBuilder.class)

.userDetailsService(userService)

.passwordEncoder(bCryptPasswordEncoder.createPasswordEncoder())

.and()

.build();

}

@Bean

public WebSecurityCustomizer webSecurityCustomizer() {

return (web) -> web.ignoring()

.requestMatchers("/css/\*\*")

.requestMatchers("/js/\*\*")

.requestMatchers("/templates/\*\*")

.requestMatchers("/webjars/\*\*");

}

}

UserMvcController.java

package ip.labwork.user.controller;

import ip.labwork.user.model.UserDto;

import ip.labwork.user.model.UserRole;

import ip.labwork.user.service.UserService;

import org.springframework.data.domain.Page;

import org.springframework.security.access.annotation.Secured;

import org.springframework.stereotype.Controller;

import org.springframework.ui.Model;

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

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

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

import java.util.List;

import java.util.stream.IntStream;

@Controller

@RequestMapping("/users")

public class UserMvcController {

private final UserService userService;

public UserMvcController(UserService userService) {

this.userService = userService;

}

@GetMapping

@Secured({UserRole.AsString.ADMIN})

public String getUsers(@RequestParam(defaultValue = "1") int page,

@RequestParam(defaultValue = "5") int size,

Model model) {

final Page<UserDto> users = userService.findAllPages(page, size)

.map(UserDto::new);

model.addAttribute("users", users);

final int totalPages = users.getTotalPages();

final List<Integer> pageNumbers = IntStream.rangeClosed(1, totalPages)

.boxed()

.toList();

model.addAttribute("pages", pageNumbers);

model.addAttribute("totalPages", totalPages);

return "users";

}

}

UserSignupMvcController.java

package ip.labwork.user.controller;

import ip.labwork.user.model.User;

import ip.labwork.user.model.UserSignupDto;

import ip.labwork.user.service.UserService;

import ip.labwork.util.validation.ValidationException;

import jakarta.validation.Valid;

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 org.springframework.web.bind.annotation.RequestMapping;

@Controller

@RequestMapping(UserSignupMvcController.SIGNUP\_URL)

public class UserSignupMvcController {

public static final String SIGNUP\_URL = "/signup";

private final UserService userService;

public UserSignupMvcController(UserService userService) {

this.userService = userService;

}

@GetMapping

public String showSignupForm(Model model) {

model.addAttribute("userDto", new UserSignupDto());

return "signup";

}

@PostMapping

public String signup(@ModelAttribute("userDto") @Valid UserSignupDto userSignupDto,

BindingResult bindingResult,

Model model) {

if (bindingResult.hasErrors()) {

model.addAttribute("errors", bindingResult.getAllErrors());

return "signup";

}

try {

final User user = userService.createUser(

userSignupDto.getLogin(), userSignupDto.getPassword(), userSignupDto.getPasswordConfirm());

return "redirect:/login?created=" + user.getLogin();

} catch (ValidationException e) {

model.addAttribute("errors", e.getMessage());

return "signup";

}

}

}

User.java

package ip.labwork.user.model;

import jakarta.persistence.\*;

import jakarta.validation.constraints.NotBlank;

import jakarta.validation.constraints.Size;

import java.util.Objects;

@Entity

@Table(name = "users")

public class User {

@Id

@GeneratedValue(strategy = GenerationType.AUTO)

private Long id;

@Column(nullable = false, unique = true, length = 64)

@NotBlank

@Size(min = 3, max = 64)

private String login;

@Column(nullable = false, length = 64)

@NotBlank

@Size(min = 6, max = 64)

private String password;

private UserRole role;

public User() {

}

public User(String login, String password) {

this(login, password, UserRole.USER);

}

public User(String login, String password, UserRole role) {

this.login = login;

this.password = password;

this.role = role;

}

public Long getId() {

return id;

}

public String getLogin() {

return login;

}

public void setLogin(String login) {

this.login = login;

}

public String getPassword() {

return password;

}

public void setPassword(String password) {

this.password = password;

}

public UserRole getRole() {

return role;

}

@Override

public boolean equals(Object o) {

if (this == o) return true;

if (o == null || getClass() != o.getClass()) return false;

User user = (User) o;

return Objects.equals(id, user.id) && Objects.equals(login, user.login);

}

@Override

public int hashCode() {

return Objects.hash(id, login);

}

}

UserDto.java

package ip.labwork.user.model;

public class UserDto {

private final long id;

private final String login;

private final UserRole role;

public UserDto(User user) {

this.id = user.getId();

this.login = user.getLogin();

this.role = user.getRole();

}

public long getId() {

return id;

}

public String getLogin() {

return login;

}

public UserRole getRole() {

return role;

}

}

UserRole.java

package ip.labwork.user.model;

import org.springframework.security.core.GrantedAuthority;

public enum UserRole implements GrantedAuthority {

ADMIN,

USER;

private static final String PREFIX = "ROLE\_";

@Override

public String getAuthority() {

return PREFIX + this.name();

}

public static final class AsString {

public static final String ADMIN = PREFIX + "ADMIN";

public static final String USER = PREFIX + "USER";

}

}

UserSignupDto.java

package ip.labwork.user.model;

import jakarta.validation.constraints.NotBlank;

import jakarta.validation.constraints.Size;

public class UserSignupDto {

@NotBlank

@Size(min = 3, max = 64)

private String login;

@NotBlank

@Size(min = 6, max = 64)

private String password;

@NotBlank

@Size(min = 6, max = 64)

private String passwordConfirm;

public String getLogin() {

return login;

}

public void setLogin(String login) {

this.login = login;

}

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;

}

}

UserRepository.java

package ip.labwork.user.repository;

import ip.labwork.user.model.User;

import org.springframework.data.jpa.repository.JpaRepository;

public interface UserRepository extends JpaRepository<User, Long> {

User findOneByLoginIgnoreCase(String login);

}

UserService.java

package ip.labwork.user.service;

import ip.labwork.user.model.User;

import ip.labwork.user.model.UserRole;

import ip.labwork.user.repository.UserRepository;

import ip.labwork.util.validation.ValidationException;

import ip.labwork.util.validation.ValidatorUtil;

import org.springframework.data.domain.Page;

import org.springframework.data.domain.PageRequest;

import org.springframework.data.domain.Sort;

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.password.PasswordEncoder;

import org.springframework.stereotype.Service;

import java.util.Collections;

import java.util.Objects;

@Service

public class UserService implements UserDetailsService {

private final UserRepository userRepository;

private final PasswordEncoder passwordEncoder;

private final ValidatorUtil validatorUtil;

public UserService(UserRepository userRepository,

PasswordEncoder passwordEncoder,

ValidatorUtil validatorUtil) {

this.userRepository = userRepository;

this.passwordEncoder = passwordEncoder;

this.validatorUtil = validatorUtil;

}

public Page<User> findAllPages(int page, int size) {

return userRepository.findAll(PageRequest.of(page - 1, size, Sort.by("id").ascending()));

}

public User findByLogin(String login) {

return userRepository.findOneByLoginIgnoreCase(login);

}

public User createUser(String login, String password, String passwordConfirm) {

return createUser(login, password, passwordConfirm, UserRole.USER);

}

public User createUser(String login, String password, String passwordConfirm, UserRole role) {

if (findByLogin(login) != null) {

throw new ValidationException(String.format("User '%s' already exists", login));

}

final User user = new User(login, passwordEncoder.encode(password), role);

validatorUtil.validate(user);

if (!Objects.equals(password, passwordConfirm)) {

throw new ValidationException("Passwords not equals");

}

return userRepository.save(user);

}

@Override

public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {

final User userEntity = findByLogin(username);

if (userEntity == null) {

throw new UsernameNotFoundException(username);

}

return new org.springframework.security.core.userdetails.User(

userEntity.getLogin(), userEntity.getPassword(), Collections.singleton(userEntity.getRole()));

}

}

**Клиент (MVC)**

default.html

<!DOCTYPE html>

<html

lang="ru"

xmlns:th="http://www.thymeleaf.org"

xmlns:layout="http://www.ultraq.net.nz/thymeleaf/layout"

xmlns:sec="http://www.thymeleaf.org/thymeleaf-extras-springsecurity6"

>

<head>

<meta charset="UTF-8" />

<title>Очень вкусно и запятая</title>

<meta name="viewport" content="width=device-width, initial-scale=1" />

<script

type="text/javascript"

src="/webjars/bootstrap/5.1.3/js/bootstrap.bundle.min.js"

></script>

<link

rel="stylesheet"

href="/webjars/bootstrap/5.1.3/css/bootstrap.min.css"

/>

<link rel="stylesheet" href="/webjars/font-awesome/6.1.0/css/all.min.css" />

<link rel="stylesheet" href="/css/style.css" />

</head>

<body class="d-flex flex-column h-100">

<div id="app">

<nav class="navbar navbar-expand-lg">

<div class="container-fluid">

<a class="navbar-brand" href="/product">

<h1 class="text-black">Очень вкусно и запятая</h1>

</a>

<button

class="navbar-toggler"

type="button"

data-bs-toggle="collapse"

data-bs-target="#navbarNav"

aria-controls="navbarNav"

aria-expanded="false"

aria-label="Toggle navigation"

>

<span class="navbar-toggler-icon"></span>

</button>

<div class="collapse navbar-collapse" id="navbarNav">

<ul class="navbar-nav me-auto mb-2 mb-lg-0" th:with="activeLink=${#ctx.springRequestContext.requestUri}" sec:authorize="isAuthenticated()">

<a class="nav-link fs-4 text-black" sec:authorize="hasRole('ROLE\_ADMIN')" href="/component" th:classappend="${#strings.equals(activeLink, '/component')} ? 'active' : ''">Компоненты</a>

<a class="nav-link fs-4 text-black" href="/product" th:classappend="${#strings.equals(activeLink, '/product')} ? 'active' : ''">Продукты</a>

<a class="nav-link fs-4 text-black" href="/order" th:classappend="${#strings.equals(activeLink, '/order')} ? 'active' : ''">Заказы</a>

<a class="nav-link fs-4 text-black" href="/order/all" th:classappend="${#strings.equals(activeLink, '/order/all')} ? 'active' : ''">История заказов</a>

<a sec:authorize="hasRole('ROLE\_ADMIN')" class="nav-link fs-4 text-black" href="/users"

th:classappend="${#strings.equals(activeLink, '/users')} ? 'active' : ''">Пользователи</a>

<a class="nav-link fs-4 text-black" href="/logout">

Выход (<span th:text="${#authentication.name}"></span>)

</a>

</ul>

</div>

</div>

</nav>

<div class="container-fluid p-0">

<div

class="container container-padding"

layout:fragment="content"

></div>

</div>

<footer

class="footer mt-auto d-flex justify-content-center align-items-center"

>

ООО "Вкусно" © 2022

</footer>

</div>

</body>

<th:block layout:fragment="scripts"> </th:block>

</html>

login.html

<!DOCTYPE html>

<html lang="en"

xmlns:layout="http://www.ultraq.net.nz/thymeleaf/layout"

layout:decorate="~{default}">

<body>

<main

class="flex-shrink-0"

style="background-color: white"

layout:fragment="content">

<div class="container container-padding mt-5">

<div th:if="${param.error}" class="alert alert-danger margin-bottom">

Пользователь не найден или пароль указан не верно

</div>

<div th:if="${param.logout}" class="alert alert-success margin-bottom">

Выход успешно произведен

</div>

<div th:if="${param.created}" class="alert alert-success margin-bottom">

Пользователь '<span th:text="${param.created}"></span>' успешно создан

</div>

<form th:action="@{/login}" method="post" class="container-padding">

<div class="mb-3">

<input type="text" name="username" id="username" class="form-control"

placeholder="Логин" required="true" autofocus="true"/>

</div>

<div class="mb-3">

<input type="password" name="password" id="password" class="form-control"

placeholder="Пароль" required="true"/>

</div>

<button type="submit" class="btn btn-success button-fixed">Войти</button>

<a class="btn btn-primary button-fixed" href="/signup">Регистрация</a>

</form>

</div>

</main>

</body>

</html>

signup.html

<!DOCTYPE html>

<html lang="en"

xmlns:layout="http://www.ultraq.net.nz/thymeleaf/layout"

layout:decorate="~{default}">

<body>

<main

class="flex-shrink-0"

style="background-color: white"

layout:fragment="content">

<div class="container container-padding mt-5">

<div th:if="${errors}" th:text="${errors}" class="margin-bottom alert alert-danger"></div>

<form action="#" th:action="@{/signup}" th:object="${userDto}" method="post">

<div class="mb-3">

<input type="text" class="form-control" th:field="${userDto.login}"

placeholder="Логин" required="true" autofocus="true" maxlength="64"/>

</div>

<div class="mb-3">

<input type="password" class="form-control" th:field="${userDto.password}"

placeholder="Пароль" required="true" minlength="6" maxlength="64"/>

</div>

<div class="mb-3">

<input type="password" class="form-control" th:field="${userDto.passwordConfirm}"

placeholder="Пароль (подтверждение)" required="true" minlength="6" maxlength="64"/>

</div>

<div class="mb-3">

<button type="submit" class="btn btn-success button-fixed">Создать</button>

<a class="btn btn-primary button-fixed" href="/login">Назад</a>

</div>

</form>

</div>

</main>

</body>

</html>

users.html

<!DOCTYPE html>

<html lang="en"

xmlns:layout="http://www.ultraq.net.nz/thymeleaf/layout"

layout:decorate="~{default}">

<body>

<main style="background-color: white" layout:fragment="content">

<div class="table-responsive">

<table class="table">

<thead>

<tr>

<th scope="col">#</th>

<th scope="col">ID</th>

<th scope="col">Логин</th>

<th scope="col">Роль</th>

</tr>

</thead>

<tbody>

<tr th:each="user, iterator: ${users}">

<th scope="row" th:text="${iterator.index} + 1"></th>

<td th:text="${user.id}"></td>

<td th:text="${user.login}" style="width: 60%"></td>

<td th:text="${user.role}" style="width: 20%"></td>

</tr>

</tbody>

</table>

</div>

<div th:if="${totalPages > 0}" class="pagination">

<span style="float: left; padding: 5px 5px;">Страницы:</span>

<a th:each="page : ${pages}"

th:href="@{/users(page=${page}, size=${users.size})}"

th:text="${page}"

th:class="${page == users.number + 1} ? active">

</a>

</div>

</main>

</body>

</html>

**Сервер (React)**

OpenAPI30Configuration.java

package ip.labwork.configuration;

import io.swagger.v3.oas.models.Components;

import io.swagger.v3.oas.models.OpenAPI;

import io.swagger.v3.oas.models.security.SecurityRequirement;

import io.swagger.v3.oas.models.security.SecurityScheme;

import ip.labwork.configuration.jwt.JwtFilter;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

@Configuration

public class OpenAPI30Configuration {

public static final String API\_PREFIX = "/api/1.0";

@Bean

public OpenAPI customizeOpenAPI() {

final String securitySchemeName = JwtFilter.TOKEN\_BEGIN\_STR;

return new OpenAPI()

.addSecurityItem(new SecurityRequirement()

.addList(securitySchemeName))

.components(new Components()

.addSecuritySchemes(securitySchemeName, new SecurityScheme()

.name(securitySchemeName)

.type(SecurityScheme.Type.HTTP)

.scheme("bearer")

.bearerFormat("JWT")));

}

}

PasswordEncoderConfiguration.java

package ip.labwork.configuration;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;

import org.springframework.security.crypto.password.PasswordEncoder;

@Configuration

public class PasswordEncoderConfiguration {

@Bean

public PasswordEncoder passwordEncoder() {

return new BCryptPasswordEncoder();

}

}

SecurityConfiguration.java

package ip.labwork.configuration;

import ip.labwork.configuration.jwt.JwtFilter;

import ip.labwork.user.controller.UserController;

import ip.labwork.user.model.UserRole;

import ip.labwork.user.service.UserService;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

import org.springframework.http.HttpMethod;

import org.springframework.security.authentication.AuthenticationManager;

import org.springframework.security.config.annotation.authentication.builders.AuthenticationManagerBuilder;

import org.springframework.security.config.annotation.web.builders.HttpSecurity;

import org.springframework.security.config.annotation.web.configuration.WebSecurityCustomizer;

import org.springframework.security.config.http.SessionCreationPolicy;

import org.springframework.security.web.SecurityFilterChain;

import org.springframework.security.web.authentication.UsernamePasswordAuthenticationFilter;

@Configuration

public class SecurityConfiguration {

private final Logger log = LoggerFactory.getLogger(SecurityConfiguration.class);

public static final String SPA\_URL\_MASK = "/{path:[^\\.]\*}";

private final UserService userService;

private final JwtFilter jwtFilter;

public SecurityConfiguration(UserService userService) {

this.userService = userService;

this.jwtFilter = new JwtFilter(userService);

createAdminOnStartup();

}

private void createAdminOnStartup() {

final String admin = "admin";

if (userService.findByLogin(admin) == null) {

log.info("Admin user successfully created");

userService.createUser(admin, admin, admin, UserRole.ADMIN);

}

}

@Bean

public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {

log.info("Creating security configuration");

http.cors()

.and()

.csrf().disable()

.sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS)

.and()

.authorizeHttpRequests()

.requestMatchers("/", SPA\_URL\_MASK).permitAll()

.requestMatchers(HttpMethod.POST, UserController.URL\_SIGNUP).permitAll()

.requestMatchers(HttpMethod.POST, UserController.URL\_LOGIN).permitAll()

.requestMatchers(HttpMethod.GET, "/users/\*").permitAll()

.requestMatchers(HttpMethod.GET, "/h2-console").permitAll()

.anyRequest()

.authenticated()

.and()

.addFilterBefore(jwtFilter, UsernamePasswordAuthenticationFilter.class)

.anonymous();

return http.build();

}

@Bean

public AuthenticationManager authenticationManager(HttpSecurity http, PasswordEncoderConfiguration bCryptPasswordEncoder)

throws Exception {

return http.getSharedObject(AuthenticationManagerBuilder.class)

.userDetailsService(userService)

.passwordEncoder(bCryptPasswordEncoder.passwordEncoder())

.and()

.build();

}

@Bean

public WebSecurityCustomizer webSecurityCustomizer() {

return (web) -> web.ignoring()

.requestMatchers(HttpMethod.OPTIONS, "/\*\*")

.requestMatchers("/\*.js")

.requestMatchers("/\*.png")

.requestMatchers("/\*.jpg")

.requestMatchers("/\*.html")

.requestMatchers("/\*.css")

.requestMatchers("/assets/\*\*")

.requestMatchers("/favicon.ico")

.requestMatchers("/.js", "/.css")

.requestMatchers("/swagger-ui/index.html")

.requestMatchers("/webjars/\*\*")

.requestMatchers("/swagger-resources/\*\*")

.requestMatchers("/v3/api-docs/\*\*");

}

}

JwtException.java

package ip.labwork.configuration.jwt;

public class JwtException extends RuntimeException {

public JwtException(Throwable throwable) {

super(throwable);

}

public JwtException(String message) {

super(message);

}

}

JwtFilter.java

package ip.labwork.configuration.jwt;

import com.fasterxml.jackson.databind.ObjectMapper;

import ip.labwork.user.service.UserService;

import jakarta.servlet.FilterChain;

import jakarta.servlet.ServletException;

import jakarta.servlet.ServletRequest;

import jakarta.servlet.ServletResponse;

import jakarta.servlet.http.HttpServletRequest;

import jakarta.servlet.http.HttpServletResponse;

import org.springframework.http.MediaType;

import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;

import org.springframework.security.core.context.SecurityContextHolder;

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

import org.springframework.util.StringUtils;

import org.springframework.web.filter.GenericFilterBean;

import java.io.IOException;

public class JwtFilter extends GenericFilterBean {

private static final String AUTHORIZATION = "Authorization";

public static final String TOKEN\_BEGIN\_STR = "Bearer ";

private final UserService userService;

public JwtFilter(UserService userService) {

this.userService = userService;

}

private String getTokenFromRequest(HttpServletRequest request) {

String bearer = request.getHeader(AUTHORIZATION);

if (StringUtils.hasText(bearer) && bearer.startsWith(TOKEN\_BEGIN\_STR)) {

return bearer.substring(TOKEN\_BEGIN\_STR.length());

}

return null;

}

private void raiseException(ServletResponse response, int status, String message) throws IOException {

if (response instanceof final HttpServletResponse httpResponse) {

httpResponse.setContentType(MediaType.APPLICATION\_JSON\_VALUE);

httpResponse.setStatus(status);

final byte[] body = new ObjectMapper().writeValueAsBytes(message);

response.getOutputStream().write(body);

}

}

@Override

public void doFilter(ServletRequest request,

ServletResponse response,

FilterChain chain) throws IOException, ServletException {

if (request instanceof final HttpServletRequest httpRequest) {

final String token = getTokenFromRequest(httpRequest);

if (StringUtils.hasText(token)) {

try {

final UserDetails user = userService.loadUserByToken(token);

final UsernamePasswordAuthenticationToken auth =

new UsernamePasswordAuthenticationToken(user, null, user.getAuthorities());

SecurityContextHolder.getContext().setAuthentication(auth);

} catch (JwtException e) {

raiseException(response, HttpServletResponse.SC\_UNAUTHORIZED, e.getMessage());

return;

} catch (Exception e) {

e.printStackTrace();

raiseException(response, HttpServletResponse.SC\_INTERNAL\_SERVER\_ERROR,

String.format("Internal error: %s", e.getMessage()));

return;

}

}

}

chain.doFilter(request, response);

}

}

JwtProperties.java

package ip.labwork.configuration.jwt;

import org.springframework.boot.context.properties.ConfigurationProperties;

import org.springframework.context.annotation.Configuration;

@Configuration

@ConfigurationProperties(prefix = "jwt", ignoreInvalidFields = true)

public class JwtProperties {

private String devToken = "";

private Boolean isDev = true;

public String getDevToken() {

return devToken;

}

public void setDevToken(String devToken) {

this.devToken = devToken;

}

public Boolean isDev() {

return isDev;

}

public void setDev(Boolean dev) {

isDev = dev;

}

}

JwtProvider.java

package ip.labwork.configuration.jwt;

import com.auth0.jwt.JWT;

import com.auth0.jwt.algorithms.Algorithm;

import com.auth0.jwt.exceptions.JWTVerificationException;

import com.auth0.jwt.interfaces.DecodedJWT;

import com.auth0.jwt.interfaces.JWTVerifier;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import org.springframework.stereotype.Component;

import org.springframework.util.StringUtils;

import java.nio.charset.StandardCharsets;

import java.security.MessageDigest;

import java.security.NoSuchAlgorithmException;

import java.time.LocalDate;

import java.time.ZoneId;

import java.util.Date;

import java.util.Optional;

import java.util.UUID;

@Component

public class JwtProvider {

private final static Logger LOG = LoggerFactory.getLogger(JwtProvider.class);

private final static byte[] HEX\_ARRAY = "0123456789ABCDEF".getBytes(StandardCharsets.US\_ASCII);

private final static String ISSUER = "auth0";

private final Algorithm algorithm;

private final JWTVerifier verifier;

public JwtProvider(JwtProperties jwtProperties) {

if (!jwtProperties.isDev()) {

LOG.info("Generate new JWT key for prod");

try {

final MessageDigest salt = MessageDigest.getInstance("SHA-256");

salt.update(UUID.randomUUID().toString().getBytes(StandardCharsets.UTF\_8));

LOG.info("Use generated JWT key for prod \n{}", bytesToHex(salt.digest()));

algorithm = Algorithm.HMAC256(bytesToHex(salt.digest()));

} catch (NoSuchAlgorithmException e) {

throw new JwtException(e);

}

} else {

LOG.info("Use default JWT key for dev \n{}", jwtProperties.getDevToken());

algorithm = Algorithm.HMAC256(jwtProperties.getDevToken());

}

verifier = JWT.require(algorithm)

.withIssuer(ISSUER)

.build();

}

private static String bytesToHex(byte[] bytes) {

byte[] hexChars = new byte[bytes.length \* 2];

for (int j = 0; j < bytes.length; j++) {

int v = bytes[j] & 0xFF;

hexChars[j \* 2] = HEX\_ARRAY[v >>> 4];

hexChars[j \* 2 + 1] = HEX\_ARRAY[v & 0x0F];

}

return new String(hexChars, StandardCharsets.UTF\_8);

}

public String generateToken(String login) {

final Date issueDate = Date.from(LocalDate.now()

.atStartOfDay(ZoneId.systemDefault())

.toInstant());

final Date expireDate = Date.from(LocalDate.now()

.plusDays(15)

.atStartOfDay(ZoneId.systemDefault())

.toInstant());

return JWT.create()

.withIssuer(ISSUER)

.withIssuedAt(issueDate)

.withExpiresAt(expireDate)

.withSubject(login)

.sign(algorithm);

}

private DecodedJWT validateToken(String token) {

try {

return verifier.verify(token);

} catch (JWTVerificationException e) {

throw new JwtException(String.format("Token verification error: %s", e.getMessage()));

}

}

public boolean isTokenValid(String token) {

if (!StringUtils.hasText(token)) {

return false;

}

try {

validateToken(token);

return true;

} catch (JwtException e) {

LOG.error(e.getMessage());

return false;

}

}

public Optional<String> getLoginFromToken(String token) {

try {

return Optional.ofNullable(validateToken(token).getSubject());

} catch (JwtException e) {

LOG.error(e.getMessage());

return Optional.empty();

}

}

}

UserController.java

package ip.labwork.user.controller;

import ip.labwork.user.model.User;

import ip.labwork.user.model.UserRole;

import ip.labwork.user.service.UserService;

import jakarta.validation.Valid;

import org.springframework.data.domain.Page;

import org.springframework.security.access.annotation.Secured;

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

import org.springframework.web.bind.annotation.\*;

import java.util.List;

import java.util.stream.IntStream;

@RestController

public class UserController {

public static final String URL\_LOGIN = "/jwt/login";

public static final String URL\_SIGNUP = "/jwt/signup";

private final UserService userService;

public UserController(UserService userService) {

this.userService = userService;

}

@PostMapping(URL\_LOGIN)

public String login(@RequestBody @Valid UserDto userDto) {

return userService.loginAndGetToken(userDto);

}

@PostMapping(URL\_SIGNUP)

public UserInfoDto signup(@RequestBody @Valid UserDto userDto) {

return userService.signupAndGetToken(userDto);

}

@GetMapping("/users/{login}")

public UserDetails getCurrentUser(@PathVariable String login) {

try {

return userService.loadUserByUsername(login);

} catch (Exception e) {

return null;

}

}

@GetMapping("/user")

public String findUser(@RequestParam("token") String token) {

UserDetails userDetails = userService.loadUserByToken(token);

User user = userService.findByLogin(userDetails.getUsername());

return user.getRole().toString();

}

@GetMapping("/users")

@Secured({UserRole.AsString.ADMIN})

public UsersPageDTO getUsers(@RequestParam(defaultValue = "1") int page,

@RequestParam(defaultValue = "5") int size) {

final Page<UserDto> users = userService.findAllPages(page, size)

.map(UserDto::new);

final int totalPages = users.getTotalPages();

final List<Integer> pageNumbers = IntStream.rangeClosed(1, totalPages)

.boxed()

.toList();

return new UsersPageDTO(users, pageNumbers, totalPages);

}

}

UserDto.java

package ip.labwork.user.controller;

import ip.labwork.user.model.User;

import ip.labwork.user.model.UserRole;

import jakarta.validation.constraints.NotEmpty;

public class UserDto {

private long id;

@NotEmpty

private String login;

@NotEmpty

private String password;

private String passwordConfirm;

private UserRole role;

public UserDto() {

}

public UserDto(User user) {

this.id = user.getId();

this.login = user.getLogin();

this.role = user.getRole();

}

public long getId() {

return id;

}

public String getLogin() {

return login;

}

public String getPassword() {

return password;

}

public String getPasswordConfirm() {

return passwordConfirm;

}

public UserRole getRole() {

return role;

}

}

UserInfoDto.java

package ip.labwork.user.controller;

import ip.labwork.user.model.UserRole;

import jakarta.validation.constraints.NotEmpty;

public class UserInfoDto {

@NotEmpty

private String token;

@NotEmpty

private String login;

@NotEmpty

private UserRole role;

public UserInfoDto(String token, String login, UserRole role) {

this.token = token;

this.login = login;

this.role = role;

}

public UserInfoDto() {

}

public String getToken() {

return token;

}

public String getLogin() {

return login;

}

public UserRole getRole() {

return role;

}

}

UsersPageDTO.java

package ip.labwork.user.controller;

import org.springframework.data.domain.Page;

import java.util.List;

public class UsersPageDTO {

private Page<UserDto> users;

private List<Integer> pageNumbers;

private int totalPages;

public UsersPageDTO(Page<UserDto> users, List<Integer> pageNumbers, int totalPages) {

this.users = users;

this.pageNumbers = pageNumbers;

this.totalPages = totalPages;

}

public Page<UserDto> getUsers() {

return users;

}

public List<Integer> getPageNumbers() {

return pageNumbers;

}

public int getTotalPages() {

return totalPages;

}

}

User.java

package ip.labwork.user.model;

import jakarta.persistence.\*;

import jakarta.validation.constraints.NotBlank;

import jakarta.validation.constraints.Size;

import java.util.Objects;

@Entity

@Table(name = "users")

public class User {

@Id

@GeneratedValue(strategy = GenerationType.AUTO)

private Long id;

@Column(nullable = false, unique = true, length = 64)

@NotBlank

@Size(min = 3, max = 64)

private String login;

@Column(nullable = false, length = 64)

@NotBlank

@Size(min = 6, max = 64)

private String password;

private UserRole role;

public User() {

}

public User(String login, String password) {

this(login, password, UserRole.USER);

}

public User(String login, String password, UserRole role) {

this.login = login;

this.password = password;

this.role = role;

}

public Long getId() {

return id;

}

public String getLogin() {

return login;

}

public void setLogin(String login) {

this.login = login;

}

public String getPassword() {

return password;

}

public void setPassword(String password) {

this.password = password;

}

public UserRole getRole() {

return role;

}

@Override

public boolean equals(Object o) {

if (this == o) return true;

if (o == null || getClass() != o.getClass()) return false;

User user = (User) o;

return Objects.equals(id, user.id);

}

@Override

public int hashCode() {

return Objects.hash(id);

}

@Override

public String toString() {

return "User{" +

"id=" + id +

", login='" + login + '\'' +

", password='" + password + '\'' +

'}';

}

}

UserRole.java

package ip.labwork.user.model;

import org.springframework.security.core.GrantedAuthority;

public enum UserRole implements GrantedAuthority {

ADMIN,

USER;

private static final String PREFIX = "ROLE\_";

@Override

public String getAuthority() {

return PREFIX + this.name();

}

public static final class AsString {

public static final String ADMIN = PREFIX + "ADMIN";

public static final String USER = PREFIX + "USER";

}

}

UserRepository.java

package ip.labwork.user.repository;

import ip.labwork.user.model.User;

import org.springframework.data.jpa.repository.JpaRepository;

public interface UserRepository extends JpaRepository<User, Long> {

User findOneByLoginIgnoreCase(String login);

}

UserExistsException.java

package ip.labwork.user.service;

public class UserExistsException extends RuntimeException {

public UserExistsException(String login) {

super(String.format("User '%s' already exists", login));

}

}

UserNotFoundException.java

package ip.labwork.user.service;

public class UserNotFoundException extends RuntimeException {

public UserNotFoundException(String login) {

super(String.format("User not found '%s'", login));

}

}

UserService.java

package ip.labwork.user.service;

import ip.labwork.configuration.jwt.JwtException;

import ip.labwork.configuration.jwt.JwtProvider;

import ip.labwork.user.controller.UserDto;

import ip.labwork.user.controller.UserInfoDto;

import ip.labwork.user.model.User;

import ip.labwork.user.model.UserRole;

import ip.labwork.user.repository.UserRepository;

import ip.labwork.util.validation.ValidationException;

import ip.labwork.util.validation.ValidatorUtil;

import org.springframework.data.domain.Page;

import org.springframework.data.domain.PageRequest;

import org.springframework.data.domain.Sort;

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.password.PasswordEncoder;

import org.springframework.stereotype.Service;

import java.util.Collections;

import java.util.Objects;

@Service

public class UserService implements UserDetailsService {

private final UserRepository userRepository;

private final PasswordEncoder passwordEncoder;

private final ValidatorUtil validatorUtil;

private final JwtProvider jwtProvider;

public UserService(UserRepository userRepository,

PasswordEncoder passwordEncoder,

ValidatorUtil validatorUtil,

JwtProvider jwtProvider) {

this.userRepository = userRepository;

this.passwordEncoder = passwordEncoder;

this.validatorUtil = validatorUtil;

this.jwtProvider = jwtProvider;

}

public Page<User> findAllPages(int page, int size) {

return userRepository.findAll(PageRequest.of(page - 1, size, Sort.by("id").ascending()));

}

public User findByLogin(String login) {

return userRepository.findOneByLoginIgnoreCase(login);

}

public User createUser(String login, String password, String passwordConfirm) {

return createUser(login, password, passwordConfirm, UserRole.USER);

}

public User createUser(String login, String password, String passwordConfirm, UserRole role) {

if (findByLogin(login) != null) {

throw new UserExistsException(login);

}

final User user = new User(login, passwordEncoder.encode(password), role);

validatorUtil.validate(user);

if (!Objects.equals(password, passwordConfirm)) {

throw new ValidationException("Passwords not equals");

}

return userRepository.save(user);

}

public String loginAndGetToken(UserDto userDto) {

final User user = findByLogin(userDto.getLogin());

if (user == null) {

throw new UserNotFoundException(userDto.getLogin());

}

if (!passwordEncoder.matches(userDto.getPassword(), user.getPassword())) {

throw new UserNotFoundException(user.getLogin());

}

return jwtProvider.generateToken(user.getLogin());

}

public UserInfoDto signupAndGetToken(UserDto userDto) {

final User user = createUser(userDto.getLogin(), userDto.getPassword(), userDto.getPasswordConfirm(), UserRole.USER);

return new UserInfoDto(jwtProvider.generateToken(user.getLogin()), user.getLogin(), UserRole.USER);

}

public UserDetails loadUserByToken(String token) throws UsernameNotFoundException {

if (!jwtProvider.isTokenValid(token)) {

throw new JwtException("Bad token");

}

final String userLogin = jwtProvider.getLoginFromToken(token)

.orElseThrow(() -> new JwtException("Token is not contain Login"));

return loadUserByUsername(userLogin);

}

@Override

public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {

final User userEntity = findByLogin(username);

if (userEntity == null) {

throw new UsernameNotFoundException(username);

}

return new org.springframework.security.core.userdetails.User(

userEntity.getLogin(), userEntity.getPassword(), Collections.singleton(userEntity.getRole()));

}

}

**Клиент (React)**

App.jsx

import "./App.css";

import {

useRoutes,

Outlet,

BrowserRouter,

Routes,

Route,

} from "react-router-dom";

import Header from "./components/common/Header";

import PrivateRoute from "./components/common/PrivateRoute";

import Footer from "./components/common/Footer";

import CatalogStudents from "./components/catalogs/CatalogStudents";

import Menu from "./components/catalogs/Menu";

import Basket from "./components/catalogs/Basket";

import History from "./components/catalogs/History";

import Registration from "./components/catalogs/Registration";

import { useState } from "react";

import Login from "./components/catalogs/Login";

import Users from "./components/catalogs/Users";

function Router(props) {

return useRoutes(props.rootRoute);

}

export default function App() {

const [product, setProduct] = useState([]);

const routes = [

{ index: true, element: <CatalogStudents /> },

{

path: "catalogs/menu",

label: "Меню",

},

{

path: "catalogs/component",

label: "Компоненты",

role: "ADMIN",

},

{

path: "catalogs/basket",

label: "Корзина",

},

{ path: "catalogs/history", label: "История" },

{

path: "catalogs/users",

label: "Пользователи",

role: "ADMIN",

},

{

path: "catalogs/registration",

label: "Регистрация",

},

{

path: "catalogs/login",

label: "Вход в систему",

},

];

const links = routes.filter((route) => route.hasOwnProperty("label"));

return (

<BrowserRouter>

<Header links={links} />

<div className="content-div">

<Routes>

<Route element={<PrivateRoute role="USER" />}>

<Route

element={<Menu product={product} setProduct={setProduct} />}

path="/catalogs/menu"

exact

/>

<Route

element={<Menu product={product} setProduct={setProduct} />}

path="\*"

/>

<Route

element={<Basket product={product} setProduct={setProduct} />}

path="/catalogs/basket"

/>

<Route element={<History />} path="/catalogs/history" />

</Route>

<Route element={<PrivateRoute role="ADMIN" />}>

<Route element={<CatalogStudents />} path="/catalogs/component" />

<Route element={<Users />} path="/catalogs/users" />

</Route>

<Route element={<Login />} path="/catalogs/login" />

<Route element={<Registration />} path="/catalogs/registration" />

</Routes>

</div>

<Footer />

</BrowserRouter>

);

}

Login.jsx

import { useState, useEffect } from "react";

import { Link, useNavigate } from 'react-router-dom';

import { useRef } from "react";

export default function Login(props) {

const [login, setLogin] = useState("");

const [password, setPassword] = useState("");

const navigate = useNavigate();

useEffect(() => {

}, []);

const loginsystem = async function (login, password) {

const requestParams = {

method: "POST",

headers: {

"Content-Type": "application/json",

},

body: JSON.stringify({login: login, password: password}),

};

const response = await fetch("http://localhost:8080/jwt/login", requestParams);

const result = await response.text();

if (response.status === 200) {

localStorage.setItem("token", result);

localStorage.setItem("user", login);

getRole(result);

} else {

localStorage.removeItem("token");

localStorage.removeItem("user");

localStorage.removeItem("role");

}

}

const getRole = async function (token) {

const requestParams = {

method: "GET",

headers: {

"Content-Type": "application/json"

}

};

const requestUrl = `http://localhost:8080/user?token=${token}`;

const response = await fetch(requestUrl, requestParams);

const result = await response.text();

localStorage.setItem("role", result);

window.dispatchEvent(new Event("storage"));

navigate("/main");

}

const loginFormOnSubmit = function (event) {

event.preventDefault();

loginsystem(login, password);

};

return (

<main className="flex-shrink-0" style={{ backgroundColor: "white" }}>

<h1 className="my-5 ms-5 ">

<b>Вход в систему</b>

</h1>

<form className="row g-3" onSubmit={loginFormOnSubmit}>

<div className="mb-3 row ms-5">

<label className="col-sm-2 col-form-label" htmlFor="login">

Логин

</label>

<div className="form-outline col-sm-10">

<input

placeholder="Логин"

className="form-control w-50"

type="text"

id="login"

name="login"

value={login}

onChange={(e) => setLogin(e.target.value)}

/>

</div>

</div>

<div className="mb-3 row ms-5">

<label className="col-sm-2 col-form-label" htmlFor="password">

Пароль

</label>

<div className="col-sm-10">

<input

placeholder="Пароль"

className="form-control w-50"

type="password"

id="password"

value={password}

onChange={(e) => setPassword(e.target.value)}

/>

</div>

</div>

<h2>

<button className="btn btn-success ms-5" style={{ color: "black" }}>

Войти

</button>

</h2>

</form>

</main>

);

}

Registration.jsx

import { useState } from "react";

import { useNavigate } from "react-router-dom";

import { useEffect } from "react";

import { Link } from 'react-router-dom';

import { useRef } from "react";

export default function Registration(props) {

const [login, setLogin] = useState("");

const [password, setPassword] = useState("");

const [passwordConfirm, setPasswordConfirm] = useState("");

const navigate = useNavigate();

useEffect(() => {}, []);

async function signup() {

const requestParams = {

method: "POST",

headers: {

"Content-Type": "application/json",

},

body: JSON.stringify({

login: login,

password: password,

passwordConfirm: passwordConfirm,

}),

};

const response = await fetch(

"http://localhost:8080/jwt/signup",

requestParams

);

const result = await response.json();

if (response.status === 200) {

localStorage.setItem("token", result.token);

localStorage.setItem("user", result.login);

localStorage.setItem("role", result.role);

window.dispatchEvent(new Event("storage"));

navigate("/catalogs/menu");

} else {

localStorage.removeItem("token");

localStorage.removeItem("user");

localStorage.removeItem("role");

alert(result);

}

}

const signupFormOnSubmit = function (event) {

event.preventDefault();

signup({

login: login,

password: password,

passwordConfirm: passwordConfirm

});

};

return (

<main className="flex-shrink-0" style={{ backgroundColor: "white" }}>

<h1 className="my-5 ms-5 ">

<b>Регистрация</b>

</h1>

<form className="row g-3" onSubmit={signupFormOnSubmit}>

<div className="mb-3 row ms-5">

<label className="col-sm-2 col-form-label" htmlFor="login">

Логин

</label>

<div className="form-outline col-sm-10">

<input

placeholder="Логин"

className="form-control w-50"

type="text"

id="login"

name="login"

value={login}

onChange={(e) => setLogin(e.target.value)}

/>

</div>

</div>

<div className="mb-3 row ms-5">

<label className="col-sm-2 col-form-label" htmlFor="password">

Пароль

</label>

<div className="col-sm-10">

<input

placeholder="Пароль"

className="form-control w-50"

type="password"

id="password"

value={password}

onChange={(e) => setPassword(e.target.value)}

/>

</div>

</div>

<div className="mb-3 row ms-5">

<label className="col-sm-2 col-form-label" htmlFor="passwordConfirm">

Пароль

</label>

<div className="col-sm-10">

<input

placeholder="Подтверждение пароля"

className="form-control w-50"

type="password"

id="passwordConfirm"

value={passwordConfirm}

onChange={(e) => setPasswordConfirm(e.target.value)}

/>

</div>

</div>

<h2>

<button className="btn btn-success ms-5" style={{ color: "black" }}>

Зарегистрироваться

</button>

</h2>

</form>

</main>

);

}

Users.jsx

import { useState } from "react";

import { useEffect } from "react";

import DataService from "../../services/DataService";

export default function Users(props) {

const [users, setUsers] = useState([]);

const [pageNumbers, setPageNumbers] = useState([]);

const [pageNumber, setPageNumber] = useState();

const usersUrl = "/users";

const host = "http://localhost:8080";

useEffect(() => {

DataService.readUsersPage(host, usersUrl, 1).then((data) => {

setUsers(data.users.content);

setPageNumbers(data.pageNumbers);

setPageNumber(1);

});

}, []);

const pageButtonOnClick = function (page) {

DataService.readUsersPage(host, usersUrl, page).then((data) => {

setUsers(data.users.content);

setPageNumber(page);

});

};

return (

<>

<main className="flex-shrink-0" style={{ backgroundColor: "white" }}>

<div className="table-shell mb-3">

<table className="table">

<thead>

<tr>

<th style={{ width: "15%" }} scope="col">

ID

</th>

<th style={{ width: "30%" }} scope="col">

Логин

</th>

<th style={{ width: "15%" }} scope="col">

Роль

</th>

</tr>

</thead>

<tbody>

{users.map((user, index) => (

<tr key={index}>

<td style={{ width: "15%" }}>{user.id}</td>

<td style={{ width: "30%" }}>{user.login}</td>

<td style={{ width: "15%" }}>{user.role}</td>

</tr>

))}

</tbody>

</table>

</div>

<div>

<p>Pages:</p>

<nav>

<ul className="pagination" style={{ backgroundColor: "white" }}>

{pageNumbers.map((number, index) => (

<li key={index}

className={`page-item ${

number === pageNumber ? "active" : ""

}`}

onClick={() => pageButtonOnClick(number)}

>

<a className="page-link" >

{number}

</a>

</li>

))}

</ul>

</nav>

</div>

</main>

</>

);

}

Card.jsx

import DataService from "../../services/DataService";

export default function Card(props) {

function edit(id) {

props.onEdit(id);

}

function remove(id) {

props.onRemove(id);

}

async function mess(id) {

let currentProduct = props.product.filter((x) => x.id == id.id);

if (currentProduct.length != 0) {

let temp = props.product.filter((x) => x.id != id.id);

currentProduct[0].count++;

temp.push(currentProduct[0]);

await props.setProduct(temp);

return;

} else {

id.count++;

props.product.push(id);

props.setProduct(props.product);

}

}

return (

<div className="temp row row-cols-1 row-cols-md-3 g-4" id="tbl-items">

{props.items.map((item) => (

<div className="col" key={item.id}>

<div className="card">

<div

className="container"

style={{ width: "100%", height: "350px" }}

>

<img

src={item["image"]}

className="img-fluid rounded mx-auto d-block"

style={{ width: "100%", height: "350px", objectFit: "contain" }}

alt="Бугер"

/>

</div>

<div className="card-body">

<h5 className="card-title text-center fs-1">{item["price"]}</h5>

{localStorage.getItem("role") == "ADMIN" && (

<>

<a

href="#"

className="btn btn-outline-dark text-center d-flex justify-content-md-center mx-5"

onClick={(e) => remove(item.id, e)}

>

Удалить

</a>

<a

href="#"

type="button"

className="btn btn-outline-dark text-center d-flex justify-content-md-center mx-5"

data-bs-toggle="modal"

data-bs-target="#staticBackdrop"

onClick={(e) => edit(item.id, e)}

>

Изменить

</a>

</>

)}

<a

type="button"

className="btn btn-outline-dark text-center d-flex justify-content-md-center mx-5"

onClick={() => mess(item)}

>

в корзину

</a>

</div>

</div>

</div>

))}

</div>

);

}

Header.jsx

import {NavLink, useNavigate} from "react-router-dom";

import {useEffect, useState} from "react";

export default function Header(props) {

const [userRole, setUserRole] = useState("");

const navigate = useNavigate();

useEffect(() => {

window.addEventListener("storage", () => {

getUserRole();

});

getUserRole();

}, []);

const getUserRole = function () {

const role = localStorage.getItem("role") || "NONE";

setUserRole(role);

};

const handlelogout = function () {

window.location.reload();

navigate("/catalogs/login");

localStorage.removeItem("role");

localStorage.removeItem("user");

localStorage.removeItem("token");

}

return (

<nav className="navbar navbar-expand-lg">

<div className="container-fluid">

<NavLink className="navbar-brand" to={"/"}>

<h1>Очень вкусно и запятая</h1>

</NavLink>

<button

className="navbar-toggler"

type="button"

data-bs-toggle="collapse"

data-bs-target="#navbarSupportedContent"

aria-controls="navbarSupportedContent"

aria-expanded="false"

aria-label="Toggle navigation"

>

<span className="navbar-toggler-icon"></span>

</button>

<div className="collapse navbar-collapse" id="navbarSupportedContent">

<ul className="navbar-nav me-auto mb-2 mb-lg-0">

{props.links.map((route) => {

if (route.role == userRole || route.role == undefined) {

return (

<li key={route.path} className="nav-item">

<NavLink className="nav-link" to={route.path}>

{route.label}

</NavLink>

</li>

);

}

})}

</ul>

</div>

<span className="col text-end">

{localStorage.getItem("role") !== null &&

<a className="nav-link" onClick={handlelogout}>

{"Выход(" + localStorage.getItem("user") + ")"}

</a>

}

</span>

</div>

</nav>

);

}

PrivateRoute.jsx

import { Outlet, Navigate, useNavigate } from "react-router-dom";

import { useEffect, useState } from "react";

export default function PrivateRoute(props) {

const navigate = useNavigate();

useEffect(() => {

window.addEventListener("storage", () => {

let token = localStorage.getItem("token");

if (token) {

getRole(token).then((role) => {

if (localStorage.getItem("role") != role) {

localStorage.removeItem("token");

localStorage.removeItem("user");

localStorage.removeItem("role");

window.dispatchEvent(new Event("storage"));

navigate("/catalog/main");

}

});

}

});

}, []);

const getRole = async function (token) {

const requestParams = {

method: "GET",

headers: {

"Content-Type": "application/json",

},

};

const requestUrl = `http://localhost:8080/user?token=${token}`;

const response = await fetch(requestUrl, requestParams);

const result = await response.text();

return result;

};

let isAllowed = false;

let userRole = localStorage.getItem("role");

if (

props.role === userRole || userRole == "ADMIN"

) {

isAllowed = true;

}

return isAllowed ? <Outlet /> : <Navigate to="/catalogs/login" />;

}

ToolbarProduct.jsx

export default function ToolbarProduct(props) {

function add() {

props.onAdd();

}

return (

<>

<div className="btn-group mt-2" role="group">

{localStorage.getItem("role") == "ADMIN" &&

<button

type="button"

className={`btn btn-outline-dark text-center d-flex justify-content-md-center mx-5 mb-3`}

onClick={add}

>

Добавить

</button>

}

</div>

</>

);

}

User.js

export default class User {

constructor(data) {

this.id = data?.id;

this.login = data?.login || "";

this.role = data?.role || "";

}

}

DataService.js

import axios from 'axios';

export default class DataService {

static dataUrlPrefix = 'http://localhost:8080';

static async readAll(url, transformer) {

const response = await fetch(this.dataUrlPrefix + url, {headers: {

"Content-Type": "application/json",

"Authorization": "Bearer " + localStorage.getItem("token")

}});

const data = await response.json();

return data.map(item => transformer(item));

}

static async readUsersPage(dataUrlPrefix, url, page) {

const response = await axios.get(dataUrlPrefix + url + `?page=${page}`,{

headers:{

"Authorization": "Bearer " + localStorage.getItem("token")

}

});

return response.data;

}

static async readUser(dataUrlPrefix, url, login){

const response = await axios.get(dataUrlPrefix + url + `/${login}`);

return response.data;

}

static async read(url, transformer) {

const response = await axios.get(this.dataUrlPrefix + url,{headers: {

"Content-Type": "application/json",

"Authorization": "Bearer " + localStorage.getItem("token")

}});

return transformer(response.data);

}

static async create(url, data) {

const requestParams = {

method: "POST",

headers: {

"Content-Type": "application/json",

},

body: JSON.stringify(data),

};

const response = await fetch(this.dataUrlPrefix + url, requestParams);

}

static async update(url, data) {

const requestParams = {

method: "PUT",

headers: {

"Content-Type": "application/json",

"Authorization": "Bearer " + localStorage.getItem("token")

},

body: JSON.stringify(data),

};

const response = await fetch(this.dataUrlPrefix + url, requestParams);

return true;

}

static async delete(url) {

const response = await axios.delete(this.dataUrlPrefix + url,{headers: {

"Content-Type": "application/json",

"Authorization": "Bearer " + localStorage.getItem("token")

}});

return response.data.id;

}

static async readUser(url, data) {

const response = await axios.get(this.dataUrlPrefix + url + `/${data}`);

return response.data;

}

static async readAllOrders(url, transformer) {

const response = await fetch(this.dataUrlPrefix + url, {headers: {

"Content-Type": "application/json",

"Authorization": "Bearer " + localStorage.getItem("token")

}});

const data = await response.json();

return data.map(item => transformer(item));

}

}