UserEntity

package study.loginstudy.domain.entity;

import lombok.AllArgsConstructor;

import lombok.Builder;

import lombok.Getter;

import lombok.NoArgsConstructor;

import study.loginstudy.domain.UserRole;

import javax.persistence.Entity;

import javax.persistence.GeneratedValue;

import javax.persistence.GenerationType;

import javax.persistence.Id;

@Entity

@Builder

@Getter

@NoArgsConstructor

@AllArgsConstructor

public class User {

@Id @GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String loginId;

private String password;

private String nickname;

private UserRole role;

}

UserRole

package study.loginstudy.domain;

public enum UserRole {

USER, ADMIN;

}

JoinRequest

package study.loginstudy.domain.dto;

import lombok.Getter;

import lombok.NoArgsConstructor;

import lombok.Setter;

import study.loginstudy.domain.UserRole;

import study.loginstudy.domain.entity.User;

import javax.validation.constraints.NotBlank;

@Getter

@Setter

@NoArgsConstructor

public class JoinRequest {

@NotBlank(message = "로그인 아이디가 비어있습니다.")

private String loginId;

@NotBlank(message = "비밀번호가 비어있습니다.")

private String password;

private String passwordCheck;

@NotBlank(message = "닉네임이 비어있습니다.")

private String nickname;

// 비밀번호 암호화 X

public User toEntity() {

return User.builder()

.loginId(this.loginId)

.password(this.password)

.nickname(this.nickname)

.role(UserRole.USER)

.build();

}

// 비밀번호 암호화

public User toEntity(String encodedPassword) {

return User.builder()

.loginId(this.loginId)

.password(encodedPassword)

.nickname(this.nickname)

.role(UserRole.USER)

.build();

}

}

LoginRequest

package study.loginstudy.domain.dto;

import lombok.Getter;

import lombok.NoArgsConstructor;

import lombok.Setter;

@Getter

@Setter

@NoArgsConstructor

public class LoginRequest {

private String loginId;

private String password;

}

UserRepository

package study.loginstudy.repository;

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

import study.loginstudy.domain.entity.User;

import java.util.Optional;

public interface UserRepository extends JpaRepository<User, Long> {

boolean existsByLoginId(String loginId);

boolean existsByNickname(String nickname);

Optional<User> findByLoginId(String loginId);

}

UserService

package study.loginstudy.service;

import lombok.RequiredArgsConstructor;

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

import org.springframework.stereotype.Service;

import org.springframework.transaction.annotation.Transactional;

import study.loginstudy.domain.dto.JoinRequest;

import study.loginstudy.domain.dto.LoginRequest;

import study.loginstudy.domain.entity.User;

import study.loginstudy.repository.UserRepository;

import java.util.Optional;

@Service

@Transactional

@RequiredArgsConstructor

public class UserService {

private final UserRepository userRepository;

private final BCryptPasswordEncoder encoder;

/\*\*

\* loginId 중복 체크

\* 회원가입 기능 구현 시 사용

\* 중복되면 true return

\*/

public boolean checkLoginIdDuplicate(String loginId) {

return userRepository.existsByLoginId(loginId);

}

/\*\*

\* nickname 중복 체크

\* 회원가입 기능 구현 시 사용

\* 중복되면 true return

\*/

public boolean checkNicknameDuplicate(String nickname) {

return userRepository.existsByNickname(nickname);

}

/\*\*

\* 회원가입 기능 1

\* 화면에서 JoinRequest(loginId, password, nickname)을 입력받아 User로 변환 후 저장

\* loginId, nickname 중복 체크는 Controller에서 진행 => 에러 메세지 출력을 위해

\*/

public void join(JoinRequest req) {

userRepository.save(req.toEntity());

}

/\*\*

\* 회원가입 기능 2

\* 화면에서 JoinRequest(loginId, password, nickname)을 입력받아 User로 변환 후 저장

\* 회원가입 1과는 달리 비밀번호를 암호화해서 저장

\* loginId, nickname 중복 체크는 Controller에서 진행 => 에러 메세지 출력을 위해

\*/

public void join2(JoinRequest req) {

userRepository.save(req.toEntity(encoder.encode(req.getPassword())));

}

/\*\*

\* 로그인 기능

\* 화면에서 LoginRequest(loginId, password)을 입력받아 loginId와 password가 일치하면 User return

\* loginId가 존재하지 않거나 password가 일치하지 않으면 null return

\*/

public User login(LoginRequest req) {

Optional<User> optionalUser = userRepository.findByLoginId(req.getLoginId());

// loginId와 일치하는 User가 없으면 null return

if(optionalUser.isEmpty()) {

return null;

}

User user = optionalUser.get();

// 찾아온 User의 password와 입력된 password가 다르면 null return

if(!user.getPassword().equals(req.getPassword())) {

return null;

}

return user;

}

/\*\*

\* userId(Long)를 입력받아 User을 return 해주는 기능

\* 인증, 인가 시 사용

\* userId가 null이거나(로그인 X) userId로 찾아온 User가 없으면 null return

\* userId로 찾아온 User가 존재하면 User return

\*/

public User getLoginUserById(Long userId) {

if(userId == null) return null;

Optional<User> optionalUser = userRepository.findById(userId);

if(optionalUser.isEmpty()) return null;

return optionalUser.get();

}

/\*\*

\* loginId(String)를 입력받아 User을 return 해주는 기능

\* 인증, 인가 시 사용

\* loginId가 null이거나(로그인 X) userId로 찾아온 User가 없으면 null return

\* loginId로 찾아온 User가 존재하면 User return

\*/

public User getLoginUserByLoginId(String loginId) {

if(loginId == null) return null;

Optional<User> optionalUser = userRepository.findByLoginId(loginId);

if(optionalUser.isEmpty()) return null;

return optionalUser.get();

}

}

CookieLoginController

package study.loginstudy.controller;

import lombok.RequiredArgsConstructor;

import org.springframework.stereotype.Controller;

import org.springframework.ui.Model;

import org.springframework.validation.BindingResult;

import org.springframework.validation.FieldError;

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

import study.loginstudy.domain.UserRole;

import study.loginstudy.domain.dto.JoinRequest;

import study.loginstudy.domain.dto.LoginRequest;

import study.loginstudy.domain.entity.User;

import study.loginstudy.service.UserService;

import javax.servlet.http.Cookie;

import javax.servlet.http.HttpServletResponse;

import javax.validation.Valid;

@Controller

@RequiredArgsConstructor

@RequestMapping("/cookie-login")

public class CookieLoginController {

private final UserService userService;

@GetMapping(value = {"", "/"})

public String home(@CookieValue(name = "userId", required = false) Long userId, Model model) {

User loginUser = userService.getLoginUserById(userId);

if(loginUser != null) {

model.addAttribute("nickname", loginUser.getNickname());

}

return "home";

}

@GetMapping("/join")

public String joinPage(Model model) {

model.addAttribute("joinRequest", new JoinRequest());

return "join";

}

@PostMapping("/join")

public String join(@Valid @ModelAttribute JoinRequest joinRequest, BindingResult bindingResult) {

// loginId 중복 체크

if(userService.checkLoginIdDuplicate(joinRequest.getLoginId())) {

bindingResult.addError(new FieldError("joinRequest", "loginId", "로그인 아이디가 중복됩니다."));

}

// 닉네임 중복 체크

if(userService.checkNicknameDuplicate(joinRequest.getNickname())) {

bindingResult.addError(new FieldError("joinRequest", "nickname", "닉네임이 중복됩니다."));

}

// password와 passwordCheck가 같은지 체크

if(!joinRequest.getPassword().equals(joinRequest.getPasswordCheck())) {

bindingResult.addError(new FieldError("joinRequest", "passwordCheck", "바밀번호가 일치하지 않습니다."));

}

if(bindingResult.hasErrors()) {

return "join";

}

userService.join(joinRequest);

return "redirect:/cookie-login";

}

@GetMapping("/login")

public String loginPage(Model model) {

model.addAttribute("loginRequest", new LoginRequest());

return "login";

}

@PostMapping("/login")

public String login(@ModelAttribute LoginRequest loginRequest, BindingResult bindingResult,

HttpServletResponse response) {

User user = userService.login(loginRequest);

// 로그인 아이디나 비밀번호가 틀린 경우 global error return

if(user == null) {

bindingResult.reject("loginFail", "로그인 아이디 또는 비밀번호가 틀렸습니다.");

}

if(bindingResult.hasErrors()) {

return "login";

}

// 로그인 성공 => 쿠키 생성

Cookie cookie = new Cookie("userId", String.valueOf(user.getId()));

cookie.setMaxAge(60 \* 60); // 쿠키 유효 시간 : 1시간

response.addCookie(cookie);

return "redirect:/cookie-login";

}

@GetMapping("/logout")

public String logout(HttpServletResponse response) {

Cookie cookie = new Cookie("userId", null);

cookie.setMaxAge(0);

response.addCookie(cookie);

return "redirect:/cookie-login";

}

@GetMapping("/info")

public String userInfo(@CookieValue(name = "userId", required = false) Long userId, Model model) {

User loginUser = userService.getLoginUserById(userId);

if(loginUser == null) {

return "redirect:/cookie-login/login";

}

model.addAttribute("user", loginUser);

return "info";

}

@GetMapping("/admin")

public String adminPage(@CookieValue(name = "userId", required = false) Long userId) {

User loginUser = userService.getLoginUserById(userId);

if(loginUser == null) {

return "redirect:/cookie-login/login";

}

if(!loginUser.getRole().equals(UserRole.ADMIN)) {

return "redirect:/cookie-login";

}

return "admin";

}

}