|  |
| --- |
| @Controller |
|  | //@RequestMapping("/user") |
|  | public class RegistrationController { |
|  | @Autowired |
|  | @Qualifier("messageSource") |
|  | private MessageSource messages; |
|  |  |
|  | private UserService userService; |
|  |  |
|  | private ApplicationEventPublisher eventPublisher; |
|  |  |
|  | private JavaMailSender mailSender; |
|  |  |
|  | private UserSecurityService userSecurityService; |
|  |  |
|  | private static final String GO\_TO\_REGISTRATION\_PAGE = "user/registration"; |
|  |  |
|  | private static final String USER\_DTO = "userDto"; |
|  |  |
|  |  |
|  | public RegistrationController(UserService userService, ApplicationEventPublisher eventPublisher, JavaMailSender mailSender, UserSecurityService userSecurityService) { |
|  | this.userService = userService; |
|  | this.eventPublisher = eventPublisher; |
|  | this.mailSender = mailSender; |
|  | this.userSecurityService = userSecurityService; |
|  | } |
|  |  |
|  |  |
|  |  |
|  | @Loggable |
|  | @GetMapping("/user/registration") |
|  | public String showRegistrationForm(Model model) { |
|  | model.addAttribute(USER\_DTO, new UserDto()); |
|  | return GO\_TO\_REGISTRATION\_PAGE; |
|  | } |
|  |  |
|  | @Loggable |
|  | @PostMapping("/user/registration") |
|  | public String registerUserAccount(@Valid UserDto userDto, |
|  | BindingResult result, |
|  | Model model, |
|  | WebRequest request) { |
|  |  |
|  | if (result.hasErrors()) { |
|  | model.addAttribute(USER\_DTO, userDto); |
|  | return GO\_TO\_REGISTRATION\_PAGE; |
|  | } |
|  |  |
|  | User registered; |
|  | try { |
|  | registered = userService.registerNewUserAccount(userDto); |
|  | } catch (EmailExistsException e) { |
|  | result.rejectValue("email", "message.regError"); |
|  | model.addAttribute(USER\_DTO, userDto); |
|  | return GO\_TO\_REGISTRATION\_PAGE; |
|  | } |
|  |  |
|  | try { |
|  | String appUrl = request.getContextPath(); |
|  | eventPublisher.publishEvent(new OnRegistrationCompleteEvent(registered, request.getLocale(), appUrl)); |
|  | } catch (Exception me) { |
|  | model.addAttribute(USER\_DTO, userDto); |
|  | return "emailError"; |
|  | } |
|  |  |
|  | return "redirect:/login"; |
|  | } |
|  |  |
|  |  |
|  | @Loggable |
|  | @GetMapping("/registrationConfirm") |
|  | public String confirmRegistration(@RequestParam("token") String token, |
|  | WebRequest request, |
|  | Model model) { |
|  |  |
|  | Locale locale = request.getLocale(); |
|  |  |
|  | VerificationToken verificationToken = userService.getVerificationToken(token); |
|  | if (verificationToken == null) { |
|  | model.addAttribute("message", messages.getMessage("auth.message.invalidToken", null, locale)); |
|  | return "redirect:/badUser.html?lang=" + locale.getLanguage(); |
|  | } |
|  |  |
|  | User user = verificationToken.getUser(); |
|  | Calendar cal = Calendar.getInstance(); |
|  | if ((verificationToken.getExpiryDate().getTime() - cal.getTime().getTime()) <= 0) { |
|  | model.addAttribute("message", messages.getMessage("auth.message.expired", null, locale)); |
|  | model.addAttribute("expired", true); |
|  | model.addAttribute("token", token); |
|  | return "redirect:/badUser.html?lang=" + locale.getLanguage(); |
|  | } |
|  |  |
|  | userService.enableUser(user); |
|  | model.addAttribute("message", messages.getMessage("message.accountVerified", null, locale)); |
|  | return "redirect:/login.html?lang=" + request.getLocale().getLanguage(); |
|  | } |
|  |  |
|  | @Loggable |
|  | @GetMapping("/user/resendRegistrationToken") |
|  | @ResponseBody |
|  | public GenericResponse resendRegistrationToken( //TODO Move Logic to Service |
|  | HttpServletRequest request, @RequestParam("token") String existingToken) { |
|  | final VerificationToken newToken = userService.generateNewVerificationToken(existingToken); |
|  | final User user = userService.getUserByVerificationToken(newToken.getToken()); |
|  | mailSender.send(constructResendVerificationTokenEmail(getAppUrl(request), request.getLocale(), newToken, user)); |
|  | return new GenericResponse(messages.getMessage("message.resendToken", null, request.getLocale())); |
|  | } |
|  |  |
|  | @Loggable |
|  | @PostMapping("/user/resetPassword") |
|  | @ResponseBody |
|  | public GenericResponse resetPassword(final HttpServletRequest request, @RequestParam("email") final String userEmail) { |
|  | final UserDto userDto = userService.getUserDtoByEmail(userEmail); |
|  | if (userDto != null) { |
|  | final String token = UUID.randomUUID().toString(); |
|  | userService.createPasswordResetTokenForUser(userDto, token); |
|  | mailSender.send(constructResetTokenEmail(getAppUrl(request), request.getLocale(), token, userDto)); |
|  | } |
|  | return new GenericResponse(messages.getMessage("message.resetPasswordEmail", null, request.getLocale())); |
|  | } |
|  |  |
|  | @Loggable |
|  | @GetMapping("/user/changePassword") |
|  | public String showChangePasswordPage(Locale locale, Model model, |
|  | @RequestParam("id") long id, @RequestParam("token") String token) { |
|  | String result = userSecurityService.validatePasswordResetToken(id, token); |
|  | if (result != null) { |
|  | model.addAttribute("message", |
|  | messages.getMessage("auth.message." + result, null, locale)); |
|  | return "redirect:/login?lang=" + locale.getLanguage(); |
|  | } |
|  | return "redirect:/updatePassword.html?lang=" + locale.getLanguage(); |
|  | } |
|  |  |
|  | @PostMapping("/user/savePassword") |
|  | @ResponseBody |
|  | public GenericResponse savePassword(Locale locale, @Valid PasswordDto passwordDto) { |
|  | User user = (User) SecurityContextHolder.getContext().getAuthentication().getPrincipal(); |
|  |  |
|  | userService.changeUserPassword(user.getId(), passwordDto.getNewPassword()); |
|  | return new GenericResponse(messages.getMessage("message.resetPasswordSuc", null, locale)); |
|  | } |
|  |  |
|  | // ============== NON-API ============ |
|  |  |
|  |  |
|  | private SimpleMailMessage constructResendVerificationTokenEmail //TODO Move to Service |
|  | (String contextPath, Locale locale, VerificationToken newToken, User user) { |
|  | String confirmationUrl = |
|  | contextPath + "/regitrationConfirm.html?token=" + newToken.getToken(); |
|  | String message = messages.getMessage("message.resendToken", null, locale); |
|  | SimpleMailMessage email = new SimpleMailMessage(); |
|  | email.setSubject("Resend Registration Token"); |
|  | email.setText(message + " rn" + confirmationUrl); |
|  | email.setFrom("twidder.bot@yandex.ru"); |
|  | email.setTo(user.getEmail()); |
|  | return email; |
|  | } |
|  |  |
|  | private SimpleMailMessage constructResetTokenEmail(final String contextPath, final Locale locale, final String token, final UserDto userDto) { |
|  | final String url = contextPath + "/user/changePassword?id=" + userDto.getId() + "&token=" + token; |
|  | final String message = messages.getMessage("message.resetPassword", null, locale); |
|  | return constructEmail("Reset Password", message + " \r\n" + url, userDto); |
|  | } |
|  |  |
|  | private String getAppUrl(HttpServletRequest request) { |
|  | return "http://" + request.getServerName() + ":" + request.getServerPort() + request.getContextPath(); |
|  | } |
|  |  |
|  | private SimpleMailMessage constructEmail(String subject, String body, UserDto userDto) { |
|  | final SimpleMailMessage email = new SimpleMailMessage(); |
|  | email.setSubject(subject); |
|  | email.setText(body); |
|  | email.setTo(userDto.getEmail()); |
|  | email.setFrom("twidder.bot@yandex.ru"); |
|  | return email; |
|  | } |
|  | } |

**Footer**

© 2022 GitHub, Inc.

**Footer navigation**

* [Terms](https://docs.github.com/en/github/site-policy/github-terms-of-service)
* [Privacy](https://docs.github.com/en/github/site-policy/github-privacy-statement)
* [Se](https://github.com/security)

[tgenman](https://github.com/tgenman)/**[internet\_shop](https://github.com/tgenman/internet_shop)**