domonkosgabor Audit Trail module (Lombiq Technologies: OCORE-23) (#6679) ✓ History

9 contributors

162 lines (141 sloc) | 6.02 KB

```csharp
using System;
using System.Collections.Generic;
using System.Text;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Mvc;
using Microsoft.Extensions.Localization;
using Microsoft.Extensions.Logging;
using OrchardCore.Entities;
using OrchardCore.Modules;
using OrchardCore.Settings;
using OrchardCore.Users.Events;
using OrchardCore.Users.Models;
using OrchardCore.Users.Services;
using OrchardCore.Users.ViewModels;

namespace OrchardCore.Users.Controllers
{
    [Feature("OrchardCore.Users.ResetPassword")]
    public class ResetPasswordController : Controller
    {
        private readonly IUserService _userService;
        private readonly UserManager<IUser> _userManager;
        private readonly ISiteService _siteService;
        private readonly IEnumerable<IPasswordRecoveryFormEvents> _passwordRecoveryFormEvents;
        private readonly ILogger _logger;
        private readonly IStringLocalizer S;

        public ResetPasswordController(
            IUserService userService,
            UserManager<IUser> userManager,
            ISiteService siteService,
            IStringLocalizer<ResetPasswordController> stringLocalizer,
            ILogger<ResetPasswordController> logger,
            IEnumerable<IPasswordRecoveryFormEvents> passwordRecoveryFormEvents)
        {
            _userService = userService;
            _userManager = userManager;
            _siteService = siteService;

            S = stringLocalizer;
            _logger = logger;
            _passwordRecoveryFormEvents = passwordRecoveryFormEvents;
        }

        [HttpGet]
        [AllowAnonymous]
        public async Task<IActionResult> ForgotPassword()
        {
            if (!(await _siteService.GetSiteSettingsAsync()).As<ResetPasswordSettings>().AllowResetPassword)
            {
                return NotFound();
            }

            return View();
        }

        [HttpPost]
        [AllowAnonymous]
        public async Task<IActionResult> ForgotPassword(ForgotPasswordViewModel model)
        {
            if (!(await _siteService.GetSiteSettingsAsync()).As<ResetPasswordSettings>().AllowResetPassword)
            {
                return NotFound();
            }

            await _passwordRecoveryFormEvents.InvokeAsync((e, modelState) => e.RecoveringPasswordAsync((key, message) => modelState.AddModelError(key, message)), ModelState, _logg

            if (TryValidateModel(model) && ModelState.IsValid)
            {
                var user = await _userService.GetForgotPasswordUserAsync(model.Email) as User;
                if (user == null || (
                        (await _siteService.GetSiteSettingsAsync()).As<RegistrationSettings>().UsersMustValidateEmail
                        && !await _userManager.IsEmailConfirmedAsync(user))
                    )
                {
                    // returns to confirmation page anyway: we don't want to let scrapers know if a username or an email exist
                    return RedirectToLocal(Url.Action("ForgotPasswordConfirmation", "ResetPassword"));
                }

                user.ResetToken = Convert.ToBase64String(Encoding.UTF8.GetBytes(user.ResetToken));
                var resetPasswordUrl = Url.Action("ResetPassword", "ResetPassword", new { code = user.ResetToken }, HttpContext.Request.Scheme);
                // send email with callback link
                await this.SendEmailAsync(user.Email, S["Reset your password"], new LostPasswordViewModel() { User = user, LostPasswordUrl = resetPasswordUrl });

                var context = new PasswordRecoveryContext(user);

                await _passwordRecoveryFormEvents.InvokeAsync((handler, context) => handler.PasswordRecoveredAsync(context), context, _logger);

                return RedirectToLocal(Url.Action("ForgotPasswordConfirmation", "ResetPassword"));
            }

            // If we got this far, something failed, redisplay form
            return View(model);
        }

        [HttpGet]
        [AllowAnonymous]
        public IActionResult ForgotPasswordConfirmation()
        {
            return View();
        }

        [HttpGet]
        [AllowAnonymous]
        public async Task<IActionResult> ResetPassword(string code = null)
        {
            if (!(await _siteService.GetSiteSettingsAsync()).As<ResetPasswordSettings>().AllowResetPassword)
            {
                return NotFound();
            }
            if (code == null)
            {
                //"A code must be supplied for password reset.";
            }
            return View(new ResetPasswordViewModel { ResetToken = code });
        }

        [HttpPost]
        [AllowAnonymous]
        [ValidateAntiForgeryToken]
        public async Task<IActionResult> ResetPassword(ResetPasswordViewModel model)
        {
            if (!(await _siteService.GetSiteSettingsAsync()).As<ResetPasswordSettings>().AllowResetPassword)
            {
                return NotFound();
            }

            await _passwordRecoveryFormEvents.InvokeAsync((e, modelState) => e.ResettingPasswordAsync((key, message) => modelState.AddModelError(key, message)), ModelState, _logge

            if (TryValidateModel(model) && ModelState.IsValid)
            {
                if (await _userService.ResetPasswordAsync(model.Email, Encoding.UTF8.GetString(Convert.FromBase64String(model.ResetToken)), model.NewPassword, (key, message) => Mo
                {
                    return RedirectToLocal(Url.Action("ResetPasswordConfirmation", "ResetPassword"));
                }
            }

            return View(model);
        }

        [HttpGet]
        [AllowAnonymous]
        public IActionResult ResetPasswordConfirmation()
        {
            return View();
        }

        private IActionResult RedirectToLocal(string returnUrl)
        {
            if (Url.IsLocalUrl(returnUrl))
            {
                return Redirect(returnUrl);
            }
            else
            {
                return Redirect("~/");
            }
        }
    }
}
```