Password reset features are an important part of any website that allows visitors to register accounts and sign in. Such features are commonly used as a recovery mechanism by users who forget their password.

When developing MVC websites, you can implement email-based password reset functionality using the **Kentico.Membership** integration package in combination with the standard ASP.NET Identity API.

Before you start working on password reset functionality, you need to perform the following:

- Integrate the Kentico membership implementation into your MVC project
- Set up authentication functionality

Both scenarios are described on the Working with users on MVC sites page.

Compatibility with Kentico password features

The built-in password features of Kentico are NOT supported by default when resetting passwords on MVC sites. This includes:

- Password reset email templates. The content of emails sent to users during the password reset process is fully controlled by the MVC application.
- Password reset settings configured in *Settings -> Security & Membership -> Passwords -> Password reset*.
  - The URL of the password reset page is determined by the route of the corresponding MVC action.
  - The validity of password reset requests depends on your application's ASP.NET Identity configuration (1 day by default).

Manual resetting of passwords through the Kentico administration is possible, even for users of MVC sites. Administrators can change passwords by editing users in the **Users** application on the **Password** tab.

- If you reset the password using the **Generate new password** button, the user receives a notification email, based on the *Membership - Changed password* email template, containing the new password.
- If you change the password manually using the **Change password** button, the user receives a notification email, based on the *Membership - Password reset confirmation* email template, informing them of the password change. This email does not contain the new password.

Password storage format

One setting that does apply to user passwords on MVC sites is the **Password format**, which you can configure within Kentico in **Settings -> Security & Membership -> Passwords**. See Setting the user password format for more information. The integration API ensures that any passwords submitted by users on your MVC website are stored in the format configured for the Kentico application.

If your Kentico application uses custom salt values when generating password hashes, you also need to set the same values for the MVC application. Check the *appSettings* section of your Kentico application's web.config for the following keys:

- **CMSPasswordSalt**
- **CMSUserSaltColumn** (obsolete key used only for backward compatibility)

If either of the keys is present, copy them to the web.config of your MVC project.

Setting up password reset functionality

To allow users to reset their passwords on your MVC site, you need to develop the required controllers and views using the membership integration API.

Configuring the password reset email settings

The password reset solution utilizes emails that are sent by the Kentico API's email engine. Configure the email settings through the administration interface of the connected Kentico application:

- Set up SMTP servers in Kentico. See Configuring SMTP servers for more information.
- You can use the Kentico **Email queue** application to monitor the emails (if the email queue is enabled).
- Set the sender address for the emails in **Settings -> System -> No-reply email address**.

Tip: To view the full code of a functional example directly in Visual Studio, download the Kentico MVC solution from GitHub and inspect the **LearningKit** project. You can also run the Learning Kit website after connecting the project to a Kentico database.

1. Create a new controller class in your MVC project or edit an existing one.
2. Prepare a property that gets an instance of the **Kentico.Membership.UserManager** class for the current request – call *HttpContext.GetOwinContext().Get<UserManager>()*.
3. Implement two actions to allow users to submit password reset requests:
   - A basic GET action that displays an email address entry form.
   - A POST action that handles sending of password reset emails to the specified address.
4. Perform the following steps within the POST action:
   a. Get the **Kentico.Membership.User** object for the specified email address – call the **UserManager.FindByEmail** method.
   b. Generate a token for the password reset link by calling the **UserManager.GeneratePasswordResetTokenAsync** method. Requires the user's ID as a parameter.
   c. Prepare the URL of the reset link, with the user ID and generated token as query string parameters.
   d. Send the password reset email by calling the **UserManager.SendEmailAsync** method. You need to specify parameters with the email subject and content (including the reset link).
5. Add another action that handles the password reset requests – validate the token and display a password reset form. Call the **UserManager.VerifyUserToken** method to verify the validity of the password reset token.

   The **VerifyUserToken** method requires the following parameters:

   - ID of the user
   - String describing the purpose of the token – *"ResetPassword"* in this case
   - Token matching the user ID

   Pass the user ID and token as query string parameters of the password reset link within the content of the sent emails.
6. Add a POST action that accepts the input of the password reset form. Call the **UserManager.ResetPassword** method to set new passwords for users.
   - You need to supply the user ID and token from the password reset request and the new password as the method's parameters.

Password reset controller example

```
 »using System;
using System.Web;
using System.Web.Mvc;
using System.Threading.Tasks;

using Microsoft.AspNet.Identity;
```

```
using Microsoft.AspNet.Identity.Owin;

using Kentico.Membership;

namespace LearningKit.Controllers
{
    public class PasswordResetController : Controller
    {
        /// <summary>
        /// Provides access to the Kentico.Membership.UserManager instance.
        /// </summary>
        public UserManager UserManager
        {
            get
            {
                return HttpContext.GetOwinContext().Get<UserManager>();
            }
        }

        /// <summary>
        /// Allows visitors to submit their email address and request a password
reset.
        /// </summary>
        public ActionResult RequestPasswordReset()
        {
            return View();
        }

        /// <summary>
        /// Generates a password reset request for the specified email address.
        /// Accepts the email address posted via the
RequestPasswordResetViewModel.
        /// </summary>
        [HttpPost]
        [ValidateAntiForgeryToken]
        public async Task<ActionResult> RequestPasswordReset
(RequestPasswordResetViewModel model)
        {
            // Validates the received email address based on the view model
            if (!ModelState.IsValid)
            {
                return View(model);
            }

            // Gets the user entity for the specified email address
            Kentico.Membership.User user = UserManager.FindByEmail(model.Email);

            if (user != null)
            {
                // Generates a password reset token for the user
                string token = await UserManager.GeneratePasswordResetTokenAsync
(user.Id);

                // Prepares the URL of the password reset link (targets the
"ResetPassword" action)
                // Fill in the name of your controller
                string resetUrl = Url.Action("ResetPassword", "PasswordReset",
new { userId = user.Id, token }, Request.Url.Scheme);

                // Creates and sends the password reset email to the user's
```

```
address
                await UserManager.SendEmailAsync(user.Id, "Password reset
request",
                    String.Format("To reset your account's password, click <a
href=\"{0}\">here</a>", resetUrl));
            }

            // Displays a view asking the visitor to check their email and click
the password reset link
            return View("CheckYourEmail");
        }

        /// <summary>
        /// Handles the links that users click in password reset emails.
        /// If the request parameters are valid, displays a form where users can
reset their password.
        /// </summary>
        public ActionResult ResetPassword(int? userId, string token)
        {
            try
            {
                // Verifies the parameters of the password reset request
                // True if the token is valid for the specified user, false if
the token is invalid or has expired
                // By default, the generated tokens are single-use and expire in
1 day
                if (UserManager.VerifyUserToken(userId.Value, "ResetPassword",
token))
                {
                    // If the password request is valid, displays the password
reset form
                    var model = new ResetPasswordViewModel
                    {
                        UserId = userId.Value,
                        Token = token
                    };

                    return View(model);
                }

                // If the password request is invalid, returns a view informing
the user
                return View("ResetPasswordInvalid");
            }
            catch (InvalidOperationException)
            {
                // An InvalidOperationException occurs if a user with the given
ID is not found
                // Returns a view informing the user that the password reset
request is not valid
                return View("ResetPasswordInvalid");
            }
        }

        /// <summary>
        /// Resets the user's password based on the posted data.
        /// Accepts the user ID, password reset token and the new password via
the ResetPasswordViewModel.
        /// </summary>
        [HttpPost]
```

```
        [ValidateAntiForgeryToken]
        [ValidateInput(false)]
        public ActionResult ResetPassword(ResetPasswordViewModel model)
        {
            // Validates the received password data based on the view model
            if (!ModelState.IsValid)
            {
                return View(model);
            }

            try
            {
                // Changes the user's password if the provided reset token is
valid
                if (UserManager.ResetPassword(model.UserId, model.Token, model.
Password).Succeeded)
                {
                    // If the password change was successful, displays a view
informing the user
                    return View("ResetPasswordSucceeded");
                }

                // Occurs if the reset token is invalid
                // Returns a view informing the user that the password reset
failed
                return View("ResetPasswordInvalid");
            }
            catch (InvalidOperationException)
            {
                // An InvalidOperationException occurs if a user with the given
ID is not found
                // Returns a view informing the user that the password reset
failed
                return View("ResetPasswordInvalid");
            }
        }
    }
}
```

7. We recommend creating view models for your password reset actions and input forms:
    - For the reset request form, the view model needs to validate and transfer the email address value.
    - For the password reset form, the view model must contain the user ID, reset token and the new password.

Password reset view model example

```
 »using System.ComponentModel;
using System.ComponentModel.DataAnnotations;

public class ResetPasswordViewModel
{
    public int UserId
    {
        get;
        set;
    }

    public string Token
    {
        get;
        set;
    }

    [DataType(DataType.Password)]
    [Required(ErrorMessage = "The Password cannot be empty.")]
    [DisplayName("Password")]
    [MaxLength(100, ErrorMessage = "The Password cannot be longer than 100
characters.")]
    public string Password
    {
        get;
        set;
    }

    [DataType(DataType.Password)]
    [DisplayName("Password confirmation")]
    [MaxLength(100, ErrorMessage = "The Password cannot be longer than 100
characters.")]
    [Compare("Password", ErrorMessage = "The entered passwords do not
match.")]
    public string PasswordConfirmation
    {
        get;
        set;
    }
}
```

8. Design the user interface for the password reset functionality on your website:
   - Add a button or link targeting the *RequestPasswordReset* action into an appropriate location on your website (for example as a "Forgotten password" link under the sign-in form).
   - Create a view for the *RequestPasswordReset* action that displays an email submission form.
   - Create a view that informs users to check their email and click a link to reset their password *(CheckYourEmail* view in the example).
   - Create a view for the *ResetPassword* action that displays a password reset form. We recommend using a strongly typed view based on your password reset view model.
   - Create views for the results of the *ResetPassword* action – for both successful and unsuccessful password resets *(ResetPasswordSucceeded* and *ResetPasswordInvalid* views in the example).

Users can now reset their passwords on your MVC site. When a user clicks the password reset button or link and submits their email address, the system sends them an email. The email contains a link (single-use with a 1-day expiration by default) that sends the user to a password reset form, where they can set a new password. The password reset form only works for users who access the URL with a valid token parameter.