

[C# и .NET](#)[Web](#)[Форум](#)[C# 5.0 и .NET 4.5](#)[WPF](#)[ТЕМЫ WPF](#)[SILVERLIGHT 5](#)[РАБОТА С БД](#)[LINQ](#)[ASP.NET](#)[WINDOWS 8/10](#)[ПРОГРАММЫ](#)

Администрирование пользователей в ASP.NET Identity

[ASP.NET](#) --- [ASP.NET Identity](#) --- [Администрирование пользователей](#)

1 В этой статье мы завершим создание инструмента администрирования пользователей, добавив возможность редактирования и удаления. В примере ниже вы можете увидеть изменения, которые я сделал в файле Views/Admin/Index.cshtml:

[Пройди тесты](#)[C# тест \(легкий\)](#)[.NET тест \(средний\)](#)

```

@using Users.Models
@model IEnumerable<AppUser>

@{
    ViewBag.Title = "ASP.NET Identity";
}

<div class="panel panel-primary">
    <div class="panel-heading">
        Учетные записи
    </div>
    <table class="table table-striped">
        <tr>
            <th>Id</th>
            <th>Имя</th>
            <th>Email</th>
            <th></th>
        </tr>
        @if (Model.Count() == 0) {
            <tr>
                <td colspan="4" class="text-center">Нет ни одной учетной записи
            </tr>
        } else {
            foreach (AppUser user in Model)
            {
                <tr>
                    <td>@user.Id</td>
                    <td>@user.UserName</td>
                    <td>@user.Email</td>
                    <td>
                        @using (Html.BeginForm("Delete", "Admin", new { id = user.Id
                        {
                            @Html.ActionLink("Изменить", "Edit", new { id = user.Id
                                new { @class = "btn btn-primary btn-xs", style = "float: right;"
                            <button class="btn btn-danger btn-xs" type="submit">Удалить
                        }
                    </td>
                </tr>
            }
        }
    </table>
        <div class="text-align: right">
            <a href="#">Пройди тесты</a>
            <a href="#">C# тест (легкий)</a>
            <a href="#">.NET тест (средний)</a>
        </div>
    <div class="text-align: right">
        @Html.ActionLink("Создать", "Create", null, new { @class = "btn btn-primary" })
    </div>

```

Удаление пользователей

Класс управления пользователями `userManager<T>` определяет *метод* `DeleteAsync()`, который принимает экземпляр класса пользователя (`IdentityUser`) и удаляет его из базы данных. В следующем примере вы можете увидеть, как я использовал этот метод в контроллере `Admin`:

```
// ...

namespace Users.Controllers
{
    public class AdminController : Controller
    {
        // ...

        [HttpPost]
        public async Task<ActionResult> Delete(string id)
        {
            AppUser user = await UserManager.FindByIdAsync(id);

            if (user != null)
            {
                IdentityResult result = await UserManager.DeleteAsync(user);
                if (result.Succeeded)
                {
                    return RedirectToAction("Index");
                }
                else
                {
                    return View("Error", result.Errors);
                }
            }
            else
            {
                return View("Error", new string[] { "Пользователь не найден" });
            }
        }

        // ...
    }
}
```

Пройди тесты C# тест (легкий) .NET тест (средний)

Метод действия Delete принимает уникальный идентификатор пользователя в качестве строкового параметра, после чего мы используем метод FindByIdAsync(), чтобы найти объект пользователя с таким идентификатором. Метод DeleteAsync() возвращает объект типа **IdentityResult**, который я проверяю также, как и в предыдущих примерах на наличие ошибок. Вы можете протестировать новую функциональность — создайте нового пользователя, а затем нажмите «Удалить» рядом с именем пользователя в представлении Index.

В данном примере мы не создавали отдельного представления Delete, поэтому для отображения ошибок создайте файл с названием Error.cshtml в папке /Views/Shared со следующей разметкой:

```
@model IEnumerable<string>
@{
    ViewBag.Title = "Ошибка";
}
<div class="alert alert-danger">
    @switch (Model.Count()) {
        case 0:
            @: Что-то пошло не так. Пожалуйста, попробуйте еще раз!
            break;
        case 1:
            @Model.First();
            break;
        default:
            @: Следующие ошибки были обнаружены:
            <ul>
                @foreach (string error in Model) {
                    <li>@error</li>
                }
            </ul>
            break;
    }
</div>
@Html.ActionLink("OK", "Index", null, new { @class = "btn btn-default" })
```

Я переместил это представление в папку Shared, т.к. оно будет использоваться также при отображении ошибок в других контроллерах, которые мы создадим позже.

Редактирование пользователей

Пройди тесты С# тест (легкий) .NET тест (средний)

Для завершения панели администратора, мы добавим возможность редактирования адреса электронной почты и пароля для пользователя. Отмечу, что

это стандартные свойства класса `IdentityUser`, позже, я покажу как можно редактировать пользовательские свойства.

В следующем примере мы добавили два новых метода действий `Edit` в контроллер `Admin`:

```
// ...

namespace Users.Controllers
{
    public class AdminController : Controller
    {
        // ...

        public async Task<ActionResult> Edit(string id)
        {
            AppUser user = await UserManager.FindByIdAsync(id);
            if (user != null)
            {
                return View(user);
            }
            else
            {
                return RedirectToAction("Index");
            }
        }

        [HttpPost]
        public async Task<ActionResult> Edit(string id, string email, string password)
        {
            AppUser user = await UserManager.FindByIdAsync(id);
            if (user != null)
            {
                user.Email = email;
                IdentityResult validEmail
                    = await UserManager.UserValidator.ValidateAsync(user);

                if (!validEmail.Succeeded)
                {
                    AddErrorsFromResult(validEmail);
                }

                IdentityResult validPass = null;
                if (password != null)
                {
                    validPass = await UserManager.PasswordValidator.ValidateAsync(user, password);
                }

                if (!validEmail.Succeeded || !validPass.Succeeded)
                {
                    AddErrorsFromResult(validEmail);
                    AddErrorsFromResult(validPass);
                }
                else
                {
                    await UserManager.UpdateAsync(user);
                    return RedirectToAction("Index");
                }
            }
            else
            {
                return RedirectToAction("Index");
            }
        }
    }
}
```

```
11 (password != string.Empty)
{
    validPass
        = await UserManager.PasswordValidator.ValidateAsync(password);

    if (validPass.Succeeded)
    {
        user.PasswordHash =
            UserManager.PasswordHasher.HashPassword(password);
    }
    else
    {
        AddErrorsFromResult(validPass);
    }
}

if ((validEmail.Succeeded && validPass == null) ||
    (validEmail.Succeeded && password != string.Empty && validPass != null))
{
    IdentityResult result = await UserManager.UpdateAsync(user);
    if (result.Succeeded)
    {
        return RedirectToAction("Index");
    }
    else
    {
        AddErrorsFromResult(result);
    }
}
else
{
    ModelState.AddModelError("", "Пользователь не найден");
}
return View(user);
}

// ...
}
```

Пройди тесты — C# тест (легкий) — .NET тест (средний)

Первый метод действия Edit обрабатывает GET-запросы с идентификатором пользователя, выполняет поиск с помощью метода FindByIdAsync() и возвращает

представление `Edit.cshtml` с возможностью редактирования если пользователь найден, иначе перенаправляет обратно на представление `Index`.

Второй метод обрабатывает POST-запросы с данными пользователя, которые передаются в качестве параметров: имя, адрес почты и пароль. В этом методе мы должны решить несколько задач, связанных с редактированием пользователей.

Первая задача — выполнить валидацию значений, которые мы получаем из формы. В данном примере мы работаем с простым объектом данных пользователя, однако, позже я покажу вам, как взаимодействовать с более сложными объектами. Итак, нам необходимо выполнить проверки данных пользователя и пароля, которые мы добавили в предыдущей статье в класс `AddUserManager`. Начинаем с проверки адреса электронной почты:

```
// ...
user.Email = email;
IdentityResult validEmail
    = await UserManager.UserValidator.ValidateAsync(user);

if (!validEmail.Succeeded)
{
    AddErrorsFromResult(validEmail);
}
// ...
```

Следующий шаг - сменить пароль, если таковой был предоставлен. ASP.NET Identity не хранит пароли в чистом виде, а использует их хэш. Поэтому, мы сначала проверяем пароль, затем генерируем его хэш и сохраняем в свойстве `PasswordHash`.

Пароли преобразуются в хэш через реализацию *интерфейса* `IPasswordHasher` (в данном случае используется свойство `AppUserManager.PasswordHasher`). Интерфейс `IPasswordHasher` определяет метод `HashPassword()`, который принимает строковый аргумент и возвращает его хэшированное значение:

Пройди тесты C# тест (легкий) .NET тест (средний)

```
// ...  
if (password != string.Empty)  
{  
    validPass  
        = await UserManager.PasswordValidator.ValidateAsync(password);  
  
    if (validPass.Succeeded)  
    {  
        user.PasswordHash =  
            UserManager.PasswordHasher.HashPassword(password);  
    }  
    else  
    {  
        AddErrorsFromResult(validPass);  
    }  
}  
// ...
```

Изменения пользовательских данных не сохраняются в базе данных вплоть до вызова метода *UpdateAsync()*:

```
// ...  
IdentityResult result = await UserManager.UpdateAsync(user);  
// ...
```

Теперь нам осталось добавить только представление для редактирования данных пользователя, которое будет выводить в текстовых полях текущие значения имени и адреса электронной почты пользователя. Добавьте в проект файл `/Views/Admin/Edit.cshtml` со следующим содержимым:

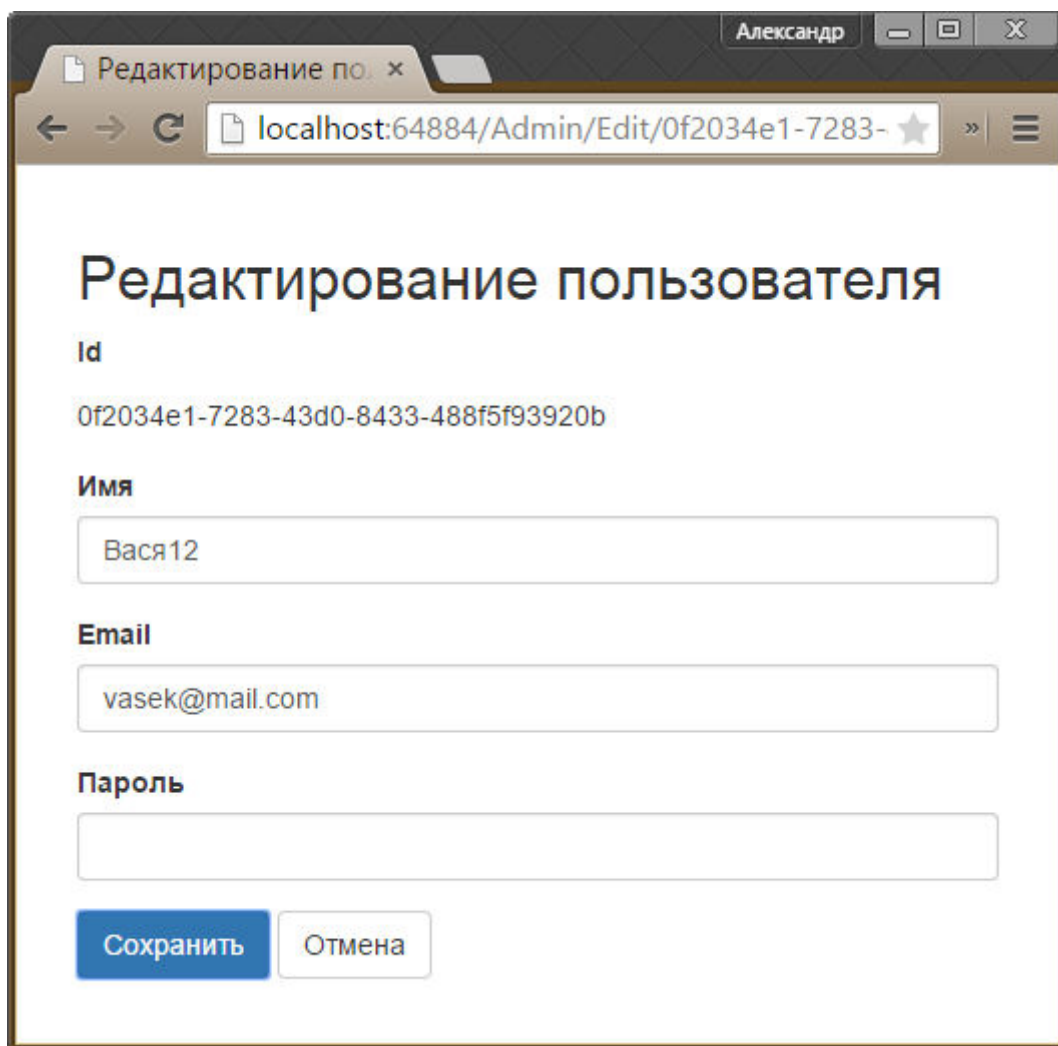

```
@model Users.Models.AppUser
@{
    ViewBag.Title = "Редактирование пользователя";
}

@Html.ValidationSummary(false)

<h2>Редактирование пользователя</h2>
<div class="form-group">
    <label>Id</label>
    <p class="form-control-static">@Model.Id</p>
</div>

@using (Html.BeginForm()) {
    @Html.HiddenFor(x => x.Id)
    <div class="form-group">
        <label>Имя</label>
        @Html.TextBoxFor(x => x.UserName, new { @class = "form-control" })
    </div>
    <div class="form-group">
        <label>Email</label>
        @Html.TextBoxFor(x => x.Email, new { @class = "form-control" })
    </div>
    <div class="form-group">
        <label>Пароль</label>
        <input name="password" type="password" class="form-control" />
    </div>
    <button type="submit" class="btn btn-primary">Сохранить</button>
    @Html.ActionLink("Отмена", "Index", null, new { @class = "btn btn-default" })
}
```

В этом представлении нет ничего необычного. Оно содержит форму со строковым идентификатором пользователя, который нельзя редактировать и поля, для ввода нового адреса электронной почты и пароля:



Александр

Редактирование по. x

localhost:64884/Admin/Edit/0f2034e1-7283-★

Редактирование пользователя

Id

0f2034e1-7283-43d0-8433-488f5f93920b

Имя

Вася12

Email

vasek@mail.com

Пароль

Сохранить Отмена

Итак, ранее мы ознакомились с тем, как нужно конфигурировать систему ASP.NET Identity и создали простое приложение администрирования пользователей с возможностью просмотра всех пользователей, добавления новых, редактирования и удаления существующих пользователей. В последующих статьях я покажу вам как настроить аутентификацию и авторизацию по ролям, использовать OAuth 2.0 и многое другое.

Alexandr Erohin ★ alexerohinzzz@gmail.com © 2011 - 2017

Пройди тесты C# тест (легкий) .NET тест (средний)

