Валидация данных в ASP.NET Identity

1 В этой статье мы продолжим проектировать наш простой проект управления пользователями и рассмотрим возможности *проверки достоверности (валидации)* введенных данных при создании новых пользователей.

Проверка паролей

Одним из наиболее распространенных требований, особенно при разработке корпоративных приложений, является контроль сложности паролей, которые используют пользователи для входа в свою учетную запись. ASP.NET Identity включает **класс PasswordValidator**, с помощью которого можно настроить систему сложности проверки паролей, используя свойства, перечисленные в таблице ниже:

Свойства, определенные в классе `PasswordValidator`

Название	Описание
RequiredLength	Задает минимально допустимую длину пароля
RequireNonLetterOrDigit	Если установлено значение `true`, пароль должен содержать хотя бы один символ, который не является ни буквой ни цифрой
RequireDigit	Если установлено значение `true`, пароль должен содержать цифры
RequireLowercase	Если установлено значение `true`, пароль должен содержать строчные символы
RequireUppercase	Если установлено значение `true`, пароль должен содержать прописные символы

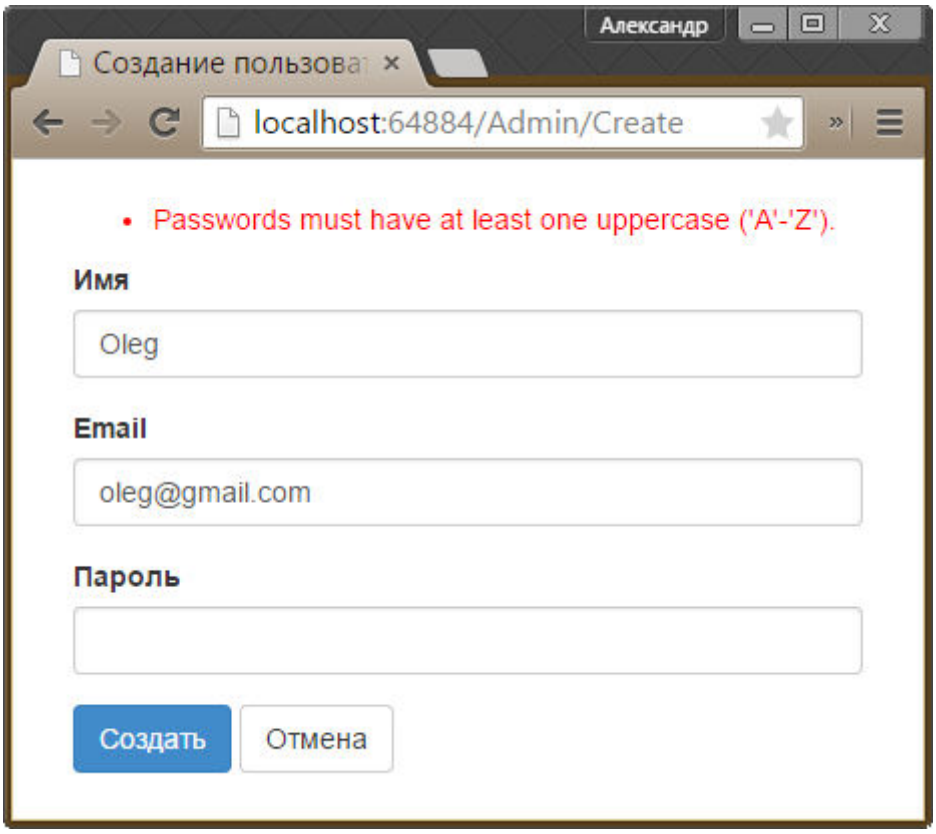
Политика определения паролей задается путем создания экземпляра класса `PasswordValidator`, установки значений его свойств и использовании этого объекта в качестве значения для свойства `PasswordValidator` класса `UserManager<T>` в методе `Create`:

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

```
using Microsoft.AspNet.Identity;
using Microsoft.AspNet.Identity.EntityFramework;
using Microsoft.AspNet.Identity.Owin;
using Microsoft.Owin;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using Users.Models;

namespace Users.Infrastructure
{
    public class AppUserManager : UserManager<AppUser>
    {
        public AppUserManager(IUserStore<AppUser> store)
            : base(store)
        { }

        public static AppUserManager Create(IdentityFactoryOptions<AppUserManager
            IOwinContext context)
        {
            AppIdentityDbContext db = context.Get<AppIdentityDbContext>();
            AppUserManager manager = new AppUserManager(new UserStore<AppUser>(db

            manager.PasswordValidator = new PasswordValidator
            {
                RequiredLength = 6,
                RequireNonLetterOrDigit = false,
                RequireDigit = false,
                RequireLowercase = true,
                RequireUppercase = true
            };

            return manager;
        }
    }
}
```

Я использовал класс PasswordValidator, чтобы определить политику, которая требует не менее шести символов, сочетани Пройди тесты C# тест (легкий) с .NET тест (средний) бы протестировать эту функциональность, запустите приложение и перейдите по адресу /Admin/Index, затем нажмите кнопку «Создать» и попытайтесь создать пользователя с

пустым паролем. При отправке формы, пароль не пройдет проверку достоверности, в результате отобразится ошибка:



Реализация пользовательской проверки паролей

Возможности встроенной проверки паролей являются достаточными для большинства приложений. Однако, возможно вам потребуется создать сложные проверки достоверности для паролей, выходящие за рамки стандартных свойств класса `PasswordValidator`. Расширить базовые возможности валидации паролей можно, создав класс, унаследованный от `PasswordValidator` и переопределяющий его *метод `ValidateAsync()`*. Добавьте файл класса `CustomPasswordValidator.cs` в папку `Infrastructure` со следующим содержимым:

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

```
using Microsoft.AspNet.Identity;
using System.Linq;
using System.Threading.Tasks;

namespace Users.Infrastructure
{
    public class CustomPasswordValidator : PasswordValidator
    {
        public override async Task<IdentityResult> ValidateAsync(string pass)
        {
            IdentityResult result = await base.ValidateAsync(pass);

            if (pass.Contains("12345"))
            {
                var errors = result.Errors.ToList();
                errors.Add("Пароль не должен содержать последовательности чис
                result = new IdentityResult(errors);
            }

            return result;
        }
    }
}
```

В этом примере мы переопределили метод `ValidateAsync()` и вызвали сначала базовую реализацию этого метода для запуска встроенной проверки. Далее мы добавили пользовательскую проверку паролей на предмет того, не содержит ли он последовательности чисел «12345». Свойство `Errors` объекта `IdentityResult` доступно только для чтения, поэтому чтобы вернуть список ошибок мы создали новый объект `IdentityResult` и объединили ошибки из базовой проверки достоверности паролей с ошибками из пользовательской проверки. Я использовал `LINQ` для соединения этих ошибок.

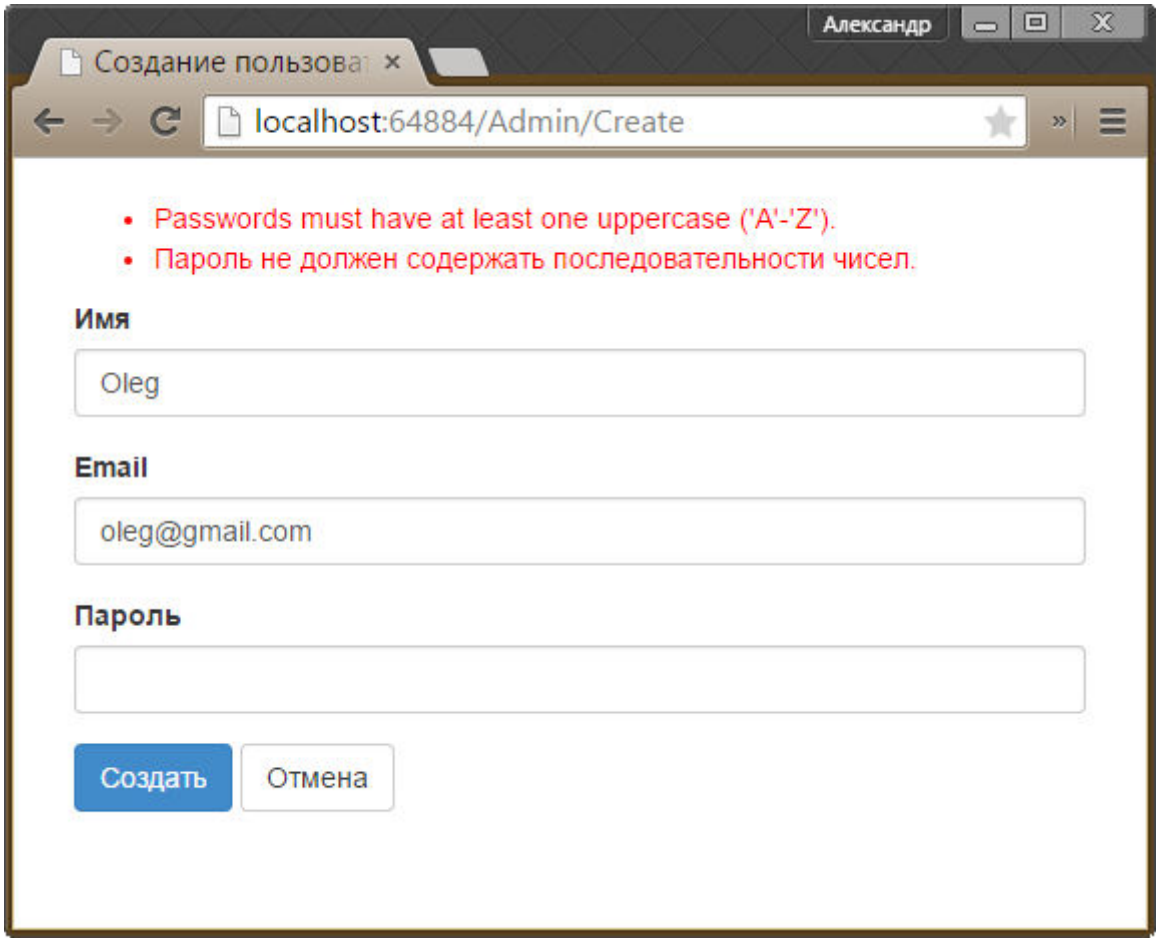
Следующий пример демонстрирует использование нового класса валидатора:

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

```
manager.PasswordValidator = new CustomPasswordValidator
{
    RequiredLength = 6,
    RequireNonLetterOrDigit = false,
    RequireDigit = false,
    RequireLowercase = true,
    RequireUppercase = true
};
```

Чтобы протестировать пользовательскую проверку паролей, попробуйте создать новую учетную запись пользователя с паролем "secret12345". Этот пароль нарушает два правила проверки — одно правило для встроенной проверки, второе — для пользовательской. Обе эти ошибки отобразятся при попытке создать нового пользователя:

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



Проверка данных пользователя

Более общие проверки могут осуществляться путем создания экземпляра класса **UserValidator** и использования его свойств из таблицы ниже:

Свойства, определенные в классе UserValidator

Название	Описание
AllowOnlyAlphanumericUserNames	Если задано true, имена пользователей могут содержать только буквы и цифры
RequireUniqueEmail	Требует наличие уникального адреса электронно почты

Выполнение проверки данных пользователя выполняется аналогично проверке паролей — свойству UserValidator класса UserManager<T> присваивается объект UserValidator, с предварительно инициализированными свойствами:

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

```
// ...

namespace Users.Infrastructure
{
    public class AppUserManager : UserManager<AppUser>
    {
        // ...

        public static AppUserManager Create(IdentityFactoryOptions<AppUserManager
            IOwinContext context)
        {
            // ...

            manager.UserValidator = new UserValidator<AppUser>(manager)
            {
                AllowOnlyAlphanumericUserNames = true,
                RequireUniqueEmail = true
            };

            return manager;
        }
    }
}
```

Класс `UserValidator` принимает обобщенный тип, который указывает на класс описания пользователя (в нашем случает это `AppUser`). В конструктор класса `UserValidator` передается экземпляр типа `UserManager`, для связывания проверки пользовательских данных с классом модели пользователя.

Также как и для проверки паролей, вы можете расширить проверку данных пользователя, создав класс, унаследованный от `UserValidator`. В качестве примера добавьте файл `CustomUserValidator.cs` в папку `Infrastructure` проекта со следующим содержимым:

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

```
using Microsoft.AspNet.Identity;
using System.Linq;
using System.Threading.Tasks;
using Users.Models;

namespace Users.Infrastructure
{
    public class CustomUserValidator : UserValidator<AppUser>
    {
        public CustomUserValidator(AppUserManager manager)
            : base(manager)
        { }

        public override async Task<IdentityResult> ValidateAsync(AppUser user)
        {
            IdentityResult result = await base.ValidateAsync(user);

            if (!user.Email.ToLower().EndsWith("@mail.com"))
            {
                var errors = result.Errors.ToList();
                errors.Add("Любой email-адрес, отличный от mail.com запрещен"
                result = new IdentityResult(errors);
            }

            return result;
        }
    }
}
```

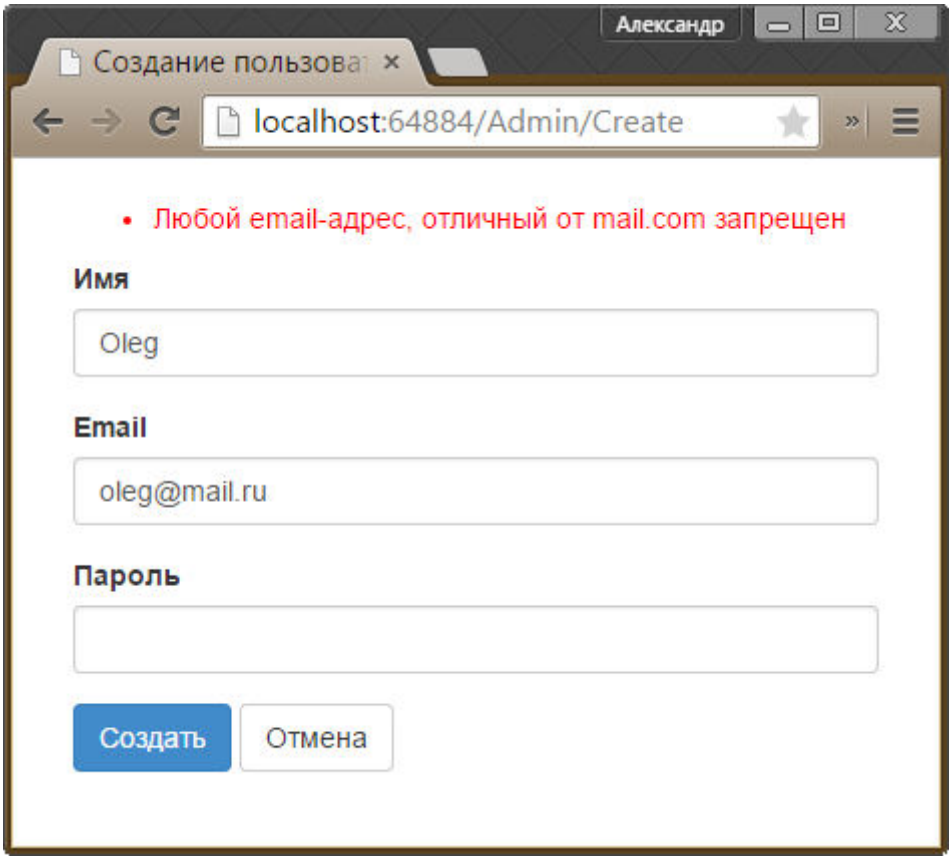
Конструктор производного класса должен принимать экземпляр класса управления пользователем (UserManager) и вызывать конструктор базового класса для запуска встроенной проверки достоверности. Настраиваемая проверка осуществляется путем переопределения метода ValidateAsync, который принимает экземпляр класса пользователя (*IdentityUser*) и возвращает объект IdentityResult. В этом примере мы добавили расширенную проверку поля адреса электронной почты, которая должна включать в себя домен «mail.com». В следующем примере мы расширили метод Create класса AppUserManager, добавив пользовательскую проверку:

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

```
// ...

namespace Users.Infrastructure
{
    public class AppUserManager : UserManager<AppUser>
    {
        // ...

        public static AppUserManager Create(IdentityFactoryOptions<AppUserManager
            IOwinContext context)
        {
            // ...

            manager.UserValidator = new CustomUserValidator(manager)
            {
                AllowOnlyAlphanumericUserNames = true,
                RequireUniqueEmail = true
            };

            return manager;
        }
    }
}
```

На рисунке ниже показан пример тестирования этой проверки, когда email-адрес пользователя не содержит домена «mail.com»:

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



Изменения текста ошибок проверки достоверности

Чуть ранее я говорил, что покажу как изменить тексты стандартных ошибок проверки достоверности. Очевидно, что ошибки на английском языке для русскоязычного приложения недопустимы. Итак, нам необходимо реализовать интерфейс валидатора *IIdentityValidator<T>*, где `T` - тип проверяемого поля. Например, рассмотренный выше класс `PasswordValidator` реализует интерфейс `IIdentityValidator<string>`, а `UserValidator` - `IIdentityValidator<IdentityUser>`. Для примера давайте изменим определение класса `CustomUserValidator`:

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

```
using Microsoft.AspNet.Identity;
using System;
using System.Collections.Generic;
using System.Text.RegularExpressions;
using System.Threading.Tasks;
using Users.Models;

namespace Users.Infrastructure
{
    public class CustomUserValidator : IIdentityValidator<AppUser>
    {
        public async Task<IdentityResult> ValidateAsync(AppUser item)
        {
            List<string> errors = new List<string>();

            if (String.IsNullOrEmpty(item.UserName.Trim()))
                errors.Add("Вы указали пустое имя.");

            string userNamePattern = @"^[a-zA-Z0-9а-яА-Я]+$";

            if (!Regex.IsMatch(item.UserName, userNamePattern))
                errors.Add("В имени разрешается указывать буквы английского и

            if (errors.Count > 0)
                return IdentityResult.Failed(errors.ToArray());

            return IdentityResult.Success;
        }
    }
}
```

В этом примере мы вручную проверяем заполнение поля и список допустимых символов в имени пользователя. Также необходимо изменить вызов вспомогательного класса валидации в методе `Create()` класса `AppUserManager`:

```
manager.UserValidator = new CustomUserValidator();
```

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