Чрез този пример ще покажем как да хешираме паролата в базата. Хеширането се прави за да не се записва паролата в явен вид. При хеширането паролата се преобразува до друг стринг, от който практически е много трудно да преобразуваме обратно в паролата. Всеки път когато се логва потребител в системата се хешира въведената парола и се сравнява с хеша от базата. Ако съвпадат – то потребителя може да влезне в системата. От съображения за сигурност, освен хеш стойността се използва и допълнителен стринг наречен salt.

Накрая към примера ще добавим и custom валидатор, който ще проверява дали паролата се състои само от букви и цифри.

=================

Създаваме празен проект File->New-> Asp.Net Web Application с името „MVCHashPasswordAndCustomValidator“ – Empty + MVC

За да запазваме регистрираните потребители ще създадем база данни с една таблица.

От SQL Server Management Studio – създаваме база данни с името „HashPasswordDB“. Създаваме и една таблица която има три колони:

CREATE TABLE [dbo].[Users] (

[ID] INT IDENTITY (1, 1) NOT NULL,

[Username] NVARCHAR (200) NOT NULL,

[PasswordHash] NVARCHAR (200) NOT NULL,

[PasswordSalt] NVARCHAR(200) NOT NULL,

CONSTRAINT [PK\_Users] PRIMARY KEY CLUSTERED ([ID] ASC)

);

===============================

Както и в по-ранни примери, когато имаме база данни е удобно да направим и един DB проект:

Добавяме към нашето приложение (solution) един проект <SQL Server Database Project> с името “HashPassword.DB”

Избираме десен бутон на проекта и оттам <Schema compare…>

След това в дясното меню избираме <Select target...> - <Database: Select Connection> и тук трябва да посочим името на сървъра и името на базата данни, например:

Server Name: .\sqlexpress

Database Name: HashPasswordDB

След като изберем базата от <Schema compare…> избираме <Switch source and target> и натискаме бутона <Compare>. Ще ни покаже, че в базата има една таблица. Избираме <Update> и ще бъде добавем скрипта Users.sql към нашия проект.

==============================

Сега ще добавим нов проект, който ще прави връзката към базата.

Към приложението добавяме нов проект -> Add new project -> Visual C#, Windows -> Class Library с името “DataAccess”.

Изтриваме автоматично създадения Class1.cs

Добавяме референция към MVCHashPasswordAndCustomValidator/References -> Add Reference -> solution - отметката на DataAccess за да е видим.

В DataAccess добавяме Entity Framework Data Model:

Десен бутон на проекта -> Add new item -> Data -> ADO.NET Entity Data Model с името “DatabaseModel”. Избираме <EF Designer from database> - Next - <New Connection>

Server name: localhost\sqlexpress

Database name: HashPasswordDB

Next -> Entity Framework 6.x -> Next -> +Tables.dbo.User, + Pluralize … , + Include foreign key… -> Finish

От файла DatabaseModel.Context.cs виждаме че контекста е с името HashPasswordDBEntities

и с него ще достъпваме базата данни.

=======================================

Добавяме нов проект към приложението Add new project -> Class Library с името “Repositories”

Нека сложим и референции към проектите.

Repositories /References -> Add Reference -> solution - отметката на DataAccess

MVCHashPasswordAndCustomValidator/References -> Add Reference -> solution - отметката на Repositories

----

Трябва да добавим и Entity Framework към проектите:

Tools -> NuGet Package Manager -> Manage NuGet Packages for Solution…

Маркираме Entity Framework и после избираме отметките за проектите Repositories и MVCHashPasswordAndCustomValidator като даваме Install за да го добавим към тях.

----

Отиваме в готовия проект „MVCHashPasswordAndCustomValidator\_Complete“ в директорията Repositories и от там копираме Helpers, IbaseRepository.cs, BaseRepository.cs и UserRepository.cs и после ги слагаме в нашия проект в директорията със същото име. От Solution Explorer от Visual Studio добавяме копираните файлове чрез <Include in project>. За да се видят трябва да дадем <Show all files>

Обръщаме внимание, че идеята за Базово репозитори е реализирана в предни примери. Разликата тук е, че контекста в Базовото репозитори е :

protected HashPasswordDBEntities Context;

за да може при необходимост да се достъпва от наследниците.

Също е добавен и метода:

protected DbSet<T> DBSet

{

get

{

return Context.Set<T>();

}

}

за да имаме достъп до дадена таблица (в случая е една).

Логиката по хеширането на паролата и използването на salt е реализирана в класа Helpers/PasswordManager. Важните методи са GeneratePasswordHash() - чрез който създаваме хеш и salt по подадена парола и метода IsPasswordMatch() чрез който проверяваме дали подадената парола съответства на хеша и salt от базата. Първия метод се ползва при регистрация, а втория при логване.

Всички действия върху даден потребител са дефинирани в UserRepository. Основната функционалност е от базовото репозитори, а останалата е допълнена с методите: GetUserByNameAndPassword(), RegisterUser() и GetUserByName().

==========================================================

Сега отиваме в web проекта „MVCHashPasswordAndCustomValidator“ и първото нещо което трябва да коригираме е да зададем коректен Connection String. Затова отиваме в проекта „DataAccess“, отваряме файла App.Config и копираме тага

<connectionStrings>

<add name="HashPasswordDBEntities" connectionString="......." />

</connectionStrings>

и го слагаме във файла Web.config в проекта „MVCHashPasswordAndCustomValidator“, примерно след затварящия таг </system.web>

------------

Друго нещо което ще ни трябва е функционалността за работа със сесия. Затова създаваме в проекта – нова директория с името „Helpers“ и в нея копираме файлът „LoginUserSession.cs“ по същия начин както беше за репозиторитата. В този пример е махнато пропъртито IsAdministrator.

-------------------

За регистрацията ще ни трябва view model, затова в директорията Models създаваме един клас „RegistrationViewModel“ с минимална валидация (за желаещите могат да добавят повече).

public class RegistrationViewModel

{

[Required]

public string Username { get; set; }

[Required]

[DataType(DataType.Password)]

public string Password { get; set; }

[Required]

[DataType(DataType.Password)]

[Compare("Password", ErrorMessage = "Passwords do not match")]

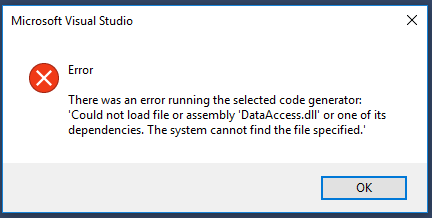
public string ConfirmPassword { get; set; }

}

------------

В Controllers добавяме нов контролер (MVC5- Empty) с името “HomeController”.

Ако не сме билднали проектите „DataAccess“ “Repositories” ще ни даде грешка че липсва съответния .dll файл.



Затова избираме от Solution Explorer дадения проект -> десен бутон и <Build>.

-----------

В HomeController създаваме Action:

[HttpGet]

public ActionResult Register()

{

return View();

}

а на този action създаваме View, като избираме да е с

Template: Edit

Model class: RegistrationViewModel

Ще видим че MVC автоматично ще добави jquery библиотеките за валидация в края на View-то. Ние ще ги преместим в Views/Shared/\_Layout.cshtml също най-отдолу (без да се повтарят).

----------

Добавяме и view за action Index() – да не е типизирано (Template: Empty) и в него слагаме само един линк:

@Html.ActionLink("Register", "Register","Home")

-----------

Сега добавяме в HomeController и един Action за Register който е по POST за да приема въведения модел.

[HttpPost]

public ActionResult Register(RegistrationViewModel viewModel)

{

if (ModelState.IsValid)

{

UserRepository userRepository = new UserRepository();

// check if the user already exists in the DB

User existingDbUser = userRepository.GetUserByName(viewModel.Username);

if (existingDbUser != null)

{

ModelState.AddModelError("", "This user is already registered in the system!");

return View();

}

User dbUser = new DataAccess.User();

dbUser.Username = viewModel.Username;

// save the user to the DB

userRepository.RegisterUser(dbUser, viewModel.Password);

TempData["Message"] = "User was registered successfully";

return RedirectToAction("Index");

}

else

{

return View();

}

}

---------------------------------------

Сега трябва да създадем и LoginViewModel в директорията Model

public class LoginViewModel

{

[Required]

public string Username { get; set; }

[Required]

//[DataType(DataType.Password)]

public string Password { get; set; }

}

След това в HomeController създаваме Action Login

[HttpGet]

public ActionResult Login()

{

return View();

}

и View, като избираме да е с Template: Edit, Model class: LoginViewModel

-------

Създаваме пак в HomeController и Action Login по POST:

[HttpPost]

public ActionResult Login(LoginViewModel viewModel)

{

if (ModelState.IsValid)

{

// here we have to check if the username exists in the database

UserRepository userRepository = new UserRepository();

User dbUser = userRepository.GetUserByNameAndPassword(viewModel.Username, viewModel.Password);

bool isUserExists = dbUser != null;

if (isUserExists)

{

LoginUserSession.Current.SetCurrentUser(dbUser.ID, dbUser.Username);

return RedirectToAction("Index");

}

else

{

ModelState.AddModelError("", "Invalid username and/or password");

}

}

// if we are here, this means there is some validation error and we have to show the login screen again

return View();

}

----------

Пробваме дали работят.

-----------

В HomeController добавяме и още един Action:

public ActionResult Logout()

{

LoginUserSession.Current.Logout();

return RedirectToAction("Index");

}

----------------

Трябва да променим Views/Shared/\_Layout.cshtml

@using MVCHashPasswordAndCustomValidator.Helpers

<!DOCTYPE html>

<html>

<head>

…

</head>

<body>

. . .

<div class="navbar-collapse collapse">

<ul class="nav navbar-nav navbar-right">

@if (LoginUserSession.Current.IsAuthenticated)

{

<li>

<span class="navbar-text">Hello @LoginUserSession.Current.Username </span>

</li>

}

<li>

@if (LoginUserSession.Current.IsAuthenticated)

{

@Html.ActionLink("Logout", "Logout", "Home")

}

else

{

@Html.ActionLink("Login", "Login", "Home")

}

</li>

</ul>

. . .

@RenderBody()

<hr />

@if (!LoginUserSession.Current.IsAuthenticated)

{

@Html.ActionLink("Register", "Register", "Home")

}

. . .

</body>

</html>

-------------

Допълваме

public ActionResult Index()

{

ViewBag.Message = TempData["Message"];

ViewBag.ErrorMessage = TempData["ErrorMessage"];

UserRepository userRepository = new UserRepository();

var allUsers = userRepository.GetAll();

return View();

}

--------------

Добавяме към Views/Shared едно ново view с име „\_InfoMessages“ със съдържание:

@\* Display the messages that added to the ViewBag

This partial view has to be somewhere at the top of the page

We create a partial view to avoid copy-paste of similar code \*@

@{

if (string.IsNullOrEmpty(ViewBag.ErrorMessage) == false)

{

<h3>

<span class="label label-danger">@ViewBag.ErrorMessage</span>

</h3>

}

if (string.IsNullOrEmpty(ViewBag.Message) == false)

{

<h3>

<span class="label label-success">@ViewBag.Message</span>

</h3>

}

}

и съответно променяме Views/Home/Index.cshtml на

@{

ViewBag.Title = "Index";

}

<h2>Index</h2>

@Html.Partial("\_InfoMessages")

====================================

Стартираме приложението и пробваме да регистрираме потребител, примерно

User: test

Pass: 123

Проверяваме в базата данни дали има записан потребител в таблицата Users. След това пробваме и логване на новия потребител.

================================

Ще направим още една функционалност. Добавяне на custom атрибут за валидация. За демострация валидатора ще проверява дали паролата съдъжа само букви и цифри.

Създаваме в MVCHashPasswordAndCustomValidator/Helpers един нов клас

[AttributeUsage(AttributeTargets.Property)]

public sealed class CustomPasswordAttribute : RequiredAttribute, IClientValidatable

{

public CustomPasswordAttribute()

{

ErrorMessage = "Password shoud contain both digits and letters";

}

public override bool IsValid(object value)

{

bool result = false;

string text = value as string;

if (string.IsNullOrEmpty(text) == false)

{

List<char> charsList = text.ToCharArray().ToList();

bool hasDigits = charsList.Any(c => char.IsDigit(c));

bool hasLetters = charsList.Any(c => char.IsLetter(c));

if (hasDigits && hasLetters)

{

result = true;

}

}

return result;

}

public IEnumerable<ModelClientValidationRule> GetClientValidationRules(ModelMetadata metadata, ControllerContext context)

{

return new ModelClientValidationRule[] {

new ModelClientValidationRule {

ValidationType = "validatepassword",

ErrorMessage = FormatErrorMessage(metadata.DisplayName)

}

};

}

}

-------------

Новия атрибут ще го използваме в LoginViewModel за полето

..

[Required]

[CustomPassword]

//[DataType(DataType.Password)]

public string Password { get; set; }

..

---------------

В MVCHashPasswordAndCustomValidator/Scripts добавяме един файл с името „jquery.validators-custom.js“ и съдържание

jQuery.validator.addMethod("validatepassword", function (value, element, param) {

var hasLetters = value.match(/[a-zA-Z]+/);

var hasDigits = value.match(/[0-9]+/);

return hasLetters && hasDigits;

});

jQuery.validator.unobtrusive.adapters.addBool("validatepassword");

-------

и трябва да добавим референция към този файл във „Views/Shared/\_Layout.cshtml“ най-отдолу

<script src="~/Scripts/jquery.validators-custom.js"></script>

Пробваме дали работи.