

در این مقاله جایگزینی پایاده سازی پیش فرض ASP.NET Identity را بررسی می‌کنیم. در ادامه خواهید خواند:

جزئیات نحوه پایاده سازی یک Storage Provider برای ASP.NET Identity

تشریح اینترفیس هایی که باید پایاده سازی شوند، و نحوه استفاده از آنها در ASP.NET Identity

ایجاد یک دیتابیس MySQL روی Windows Azure

نحوه استفاده از یک ابزار کلاینت (MySQL Workbench) برای مدیریت دیتابیس مذکور

نحوه جایگزینی پایاده سازی سفارشی با نسخه پیش فرض در یک اپلیکیشن ASP.NET MVC

در انتهای این مقاله یک اپلیکیشن ASP.NET MVC خواهیم داشت که از ASP.NET Identity و تامین کننده سفارشی جدید استفاده می‌کند. دیتابیس اپلیکیشن MySQL خواهد بود و روی Windows Azure میزبانی می‌شود. سورس کد کامل این مثال را هم می‌توانید از [این لینک](#) دریافت کنید.

پایاده سازی یک Storage Provider سفارشی برای ASP.NET Identity

ASP.NET Identity سیستم توسعه پذیری است که می‌توانید بخش‌های مختلف آن را جایگزین کنید. در این سیستم بناهای سطح بالایی مانند Managers و Stores وجود دارند.

Managers کلاس‌های سطح بالایی هستند که توسعه دهندگان از آنها برای اجرای عملیات مختلف روی ASP.NET Identity استفاده می‌کنند. مدیریت کننده‌های موجود عبارتند از UserManager و RoleManager. کلاس UserManager برای اجرای عملیات مختلف روی کاربران استفاده می‌شود، مثلاً ایجاد کاربر جدید یا حذف آنها. کلاس RoleManager هم برای اجرای عملیات مختلف روی نقش‌ها استفاده می‌شود.

Stores کلاس‌های سطح پایین‌تری هستند که جزئیات پایاده سازی را در بر می‌گیرند، مثلاً اینکه موجودیت‌های کاربران و نقش‌ها چگونه باید ذخیره و بازیابی شوند. این کلاس‌ها با مکانیزم ذخیره و بازیابی تلفیق شده اند. مثلاً

Microsoft.AspNet.Identity.EntityFramework کلاسی با نام UserStore دارد که برای ذخیره و بازیابی Userها و داده‌های مربوطه توسط EntityFramework استفاده می‌شود.

Managers از Stores تفکیک شده اند و هیچ وابستگی ای به یکدیگر ندارند. این تفکیک بدین منظور انجام شده که بتوانید مکانیزم ذخیره و بازیابی را جایگزین کنید، بدون اینکه اپلیکیشن شما از کار بیافتد یا نیاز به توسعه بیشتر داشته باشد. کلاس‌های Manager می‌توانند با هر Store ای ارتباط برقرار کنند. از آنجا که شما از APIهای سطح بالای UserManager برای انجام عملیات CRUD روی کاربران استفاده می‌کنید، اگر UserStore را با پایاده سازی دیگری جایگزین کنید، مثلاً AzureTable Storage یا MySql، نیازی به بازنویسی اپلیکیشن نیست.

در مثال جاری پایاده سازی پیش فرض Entity Framework را با یک تامین کننده MySQL جایگزین می‌کنیم.

پایاده سازی کلاس‌های Storage

برای پایاده سازی تامین کننده‌های سفارشی، باید کلاس هایی را پایاده سازی کنید که همتای آنها در

Microsoft.AspNet.Identity.EntityFramework وجود دارند:

UserStore<TUser>

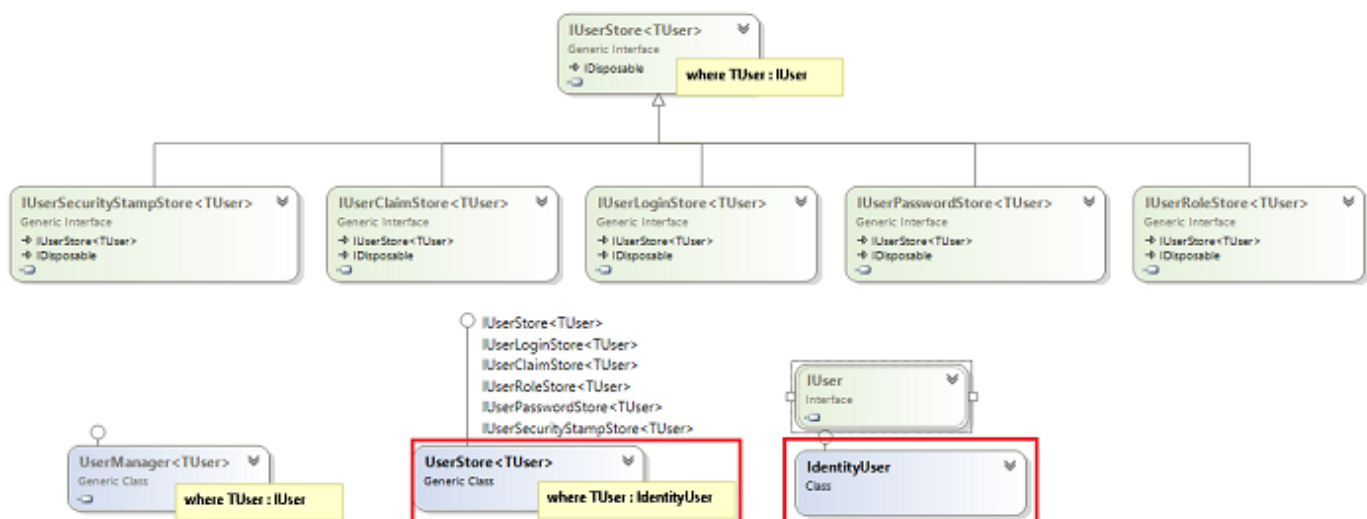
IdentityUser

RoleStore<TRole>

IdentityRole

پایاده سازی پیش فرض Entity Framework را در تصاویر زیر مشاهده می‌کنید.

Users



Roles



در مخزن پیش فرض ASP.NET Identity EntityFramework کلاس‌های بیشتری برای موجودیت‌ها مشاهده می‌کنید.

IdentityUserClaim

IdentityUserLogin

IdentityUserRole

همانطور که از نام این کلاس‌ها مشخص است، اختیارات، نقش‌ها و اطلاعات ورود کاربران توسط این کلاس‌ها معرفی می‌شوند. در مثال جاری این کلاس‌ها را پیاده سازی نخواهیم کرد، چرا که بارگذاری اینگونه رکوردها از دیتابیس به حافظه برای انجام عملیات پایه (مانند افزودن و حذف اختیارات کاربران) سنگین است. در عوض کلاس‌های backend store اینگونه عملیات را بصورت مستقیم روی دیتابیس اجرا خواهند کرد. بعنوان نمونه متد `UserStore.GetClaimsAsync()` را در نظر بگیرید. این متد به نوبه خود متد `userClaimTable.FindById(user.Id)` را فراخوانی می‌کند که یک کوئری روی جدول مربوطه اجرا می‌کند و لیستی از اختیارات کاربر را بر می‌گرداند.

```
public Task<IList<Claim>> GetClaimsAsync(IdentityUser user)
{
    ClaimsIdentity identity = userClaimsTable.FindById(user.Id);
    return Task.FromResult<IList<Claim>>(identity.Claims.ToList());
}
```

```
}
```

برای پیاده سازی یک تامین کننده سفارشی MySQL مراحل زیر را دنبال کنید.
1. کلاس کاربر را ایجاد کنید، که اینترفیس **IUser** را پیاده سازی می کند.

```
public class IdentityUser : IUser
{
    public IdentityUser(){...}
    public IdentityUser(string userName) (){...}
    public string Id { get; set; }
    public string Username { get; set; }
    public string PasswordHash { get; set; }
    public string SecurityStamp { get; set; }
}
```

2. کلاس User Store را ایجاد کنید، که اینترفیس های **IUserStore** , **IUserClaimStore** , **IUserLoginStore** , **IUserRoleStore** و **IUserPasswordStore** را پیاده سازی می کند. توجه کنید که تنها اینترفیس **IUserStore** را باید پیاده سازی کنید، مگر آنکه بخواهید از امکاناتی که دیگر اینترفیس ها ارائه می کنند هم استفاده کنید.

```
public class UserStore : IUserStore<IdentityUser>,
                        IUserClaimStore<IdentityUser>,
                        IUserLoginStore<IdentityUser>,
                        IUserRoleStore<IdentityUser>,
                        IUserPasswordStore<IdentityUser>
{
    public UserStore(){...}
    public Task CreateAsync(IdentityUser user){...}
    public Task<IdentityUser> FindByIdAsync(string userId){...}
    ...
}
```

3. کلاس Role را ایجاد کنید که اینترفیس **IRole** را پیاده سازی می کند.

```
public class IdentityRole : IRole
{
    public IdentityRole(){...}
    public IdentityRole(string roleName) (){...}
    public string Id { get; set; }
    public string Name { get; set; }
}
```

4. کلاس Role Store را ایجاد کنید که اینترفیس **IRoleStore** را پیاده سازی می کند. توجه داشته باشید که پیاده سازی این مخزن اختیاری است و در صورتی لازم است که بخواهید از نقش ها در سیستم خود استفاده کنید.

```
public class RoleStore : IRoleStore<IdentityRole>
{
    public RoleStore(){...}
    public Task CreateAsync(IdentityRole role){...}
    public Task<IdentityRole> FindByIdAsync(string roleId){...}
    ....
}
```

کلاس های بیشتری هم وجود دارند که مختص پیاده سازی مثال جاری هستند.

MySQLDatabase: این کلاس اتصال دیتابیس MySQL و کوئری‌ها را کپسوله می‌کند. کلاس‌های UserStore و RoleStore توسط نمونه ای از این کلاس و هله سازی می‌شوند.

RoleTable: این کلاس جدول Roles و عملیات CRUD مربوط به آن را کپسوله می‌کند.

UserClaimsTable: این کلاس جدول UserClaims و عملیات CRUD مربوط به آن را کپسوله می‌کند.

UserLoginsTable: این کلاس جدول UserLogins و عملیات CRUD مربوط به آن را کپسوله می‌کند.

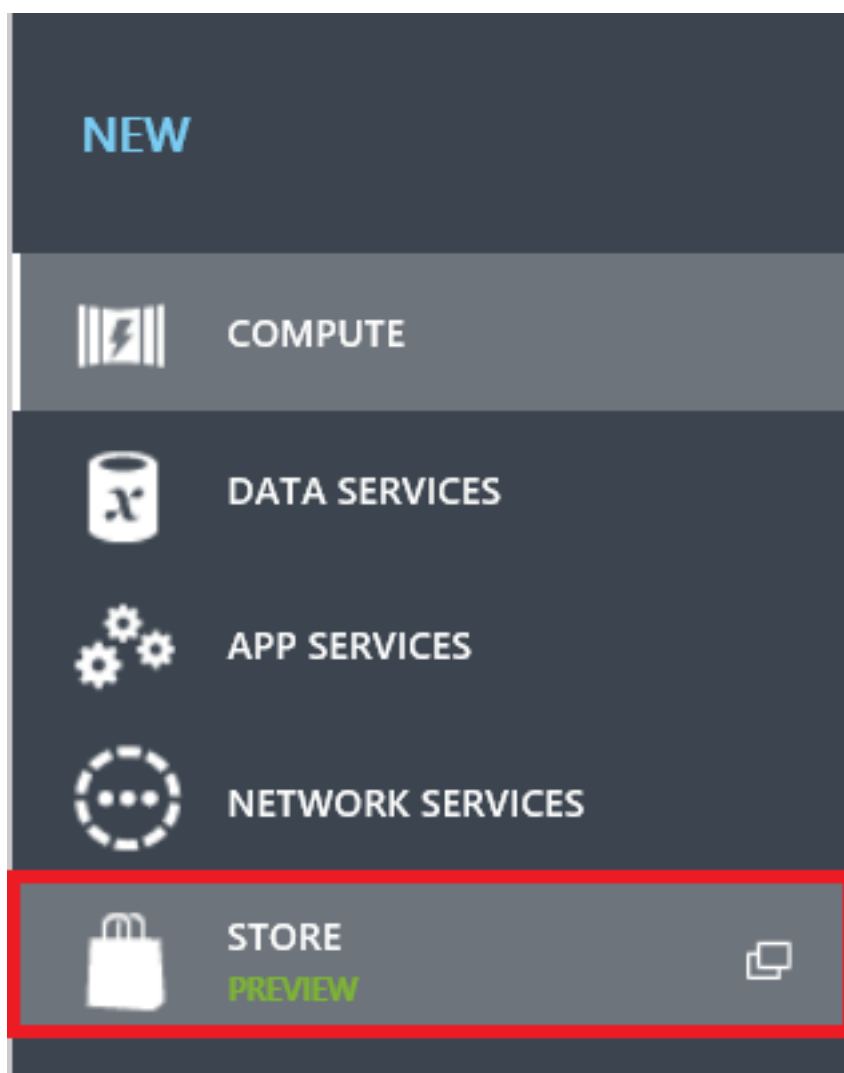
UserRolesTable: این کلاس جدول UserRoles و عملیات CRUD مربوطه به آن را کپسوله می‌کند.

UserTable: این کلاس جدول Users و عملیات CRUD مربوط به آن را کپسوله می‌کند.

ایجاد یک دیتابیس MySQL روی Windows Azure

1. به [پورتال مدیریتی Windows Azure](#) وارد شوید.

2. در پایین صفحه روی **+NEW** کلیک کنید و گزینه **STORE** را انتخاب نمایید.



در ویزارد **Choose Add-on** به سمت پایین اسکرول کنید و گزینه **ClearDB MySQL Database** را انتخاب کنید. سپس به مرحله بعد بروید.

Choose an Add-on

ALL

APP SERVICES

DATA



ClearDB MySQL Database



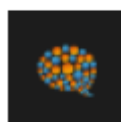
ClearPointe Azure Management



cloudinary




D&B Business Insight




Embarke Email Analytics



Engine Yard Platform as a Service






ClearDB MySQL Database

SuccessBricks, Inc.

ClearDB is a powerful, fault-tolerant database-as-a-service in the cloud for your MySQL powered applications.

PUBLISHED DATE

10/10/2012



2

3

4. راهکار **Free** بصورت پیش فرض انتخاب شده، همین گزینه را انتخاب کنید و نام دیتابیس را به **IdentityMySQLDatabase** تغییر دهید. نزدیک ترین ناحیه (region) به خود را انتخاب کنید و به مرحله بعد بروید.

PURCHASE FROM STORE

Personalize Add-on

PLANS (4)

☒ **Free**

Great for getting started and developing your apps.
Includes 20 MB of storage and up to 4 connections.

0 USD/month

☐ **Venus**

Excellent for light test and staging apps that need a reliable MySQL database. Includes support for up to 1 GB of storage and up to 15 connections.

9.99 USD/month

PROMOTION CODE

?

NAME

✓

REGION

▼

5. روی علامت checkmark کلیک کنید تا دیتابیس شما ایجاد شود. پس از آنکه دیتابیس شما ساخته شد می توانید از قسمت **ADD-**

The screenshot shows the Azure portal interface. On the left sidebar, the 'ADD-ONS' category is highlighted with a red box. The main content area displays the 'add-ons' section with a 'PREVIEW' label. Below this is a table with the following data:

NAME	TYPE
IdentityMySQLDatabase	App Service

The 'IdentityMySQLDatabase' entry is highlighted with a red box. At the bottom of the page, the 'CONNECTION INFO' button is also highlighted with a red box.

6. همانطور که در تصویر بالا می بینید، می توانید اطلاعات اتصال دیتابیس (connection info) را از پایین صفحه دریافت کنید.


7. اطلاعات اتصال را با کلیک کردن روی دکمه مجاور کپی کنید تا بعداً در اپلیکیشن MVC خود از آن استفاده کنیم.

×

Connection info


CONNECTIONSTRING


Database=IdentityMySQLDatabase;Data Source=us-cdb-azure-west-b.cleardb.com;User Id=root@us-cdb-azure-west-b.cleardb.com;Password=us-cdb-azure-west-b.cleardb.com;Persist Security Info=True;Server=mysql;SslMode=REQUIRED;TrustServerCertificate=True



CONNECTIONURL

mysql://bc554a1b496531:a7277b9f@us-cdb-azure-west-b.cleardb.com:3306/IdentityMySQLDatabase





ایجاد جداول در یک دیتابیس MySQL

ابتدا ابزار MySQL Workbench را نصب کنید.

1. ابزار مذکور را [از اینجا](#) دانلود کنید.

2. هنگام نصب، گزینه **Setup Type: Custom** را انتخاب کنید.

3. در قسمت انتخاب قابلیت ها، گزینه های **Applications** و **MySQLWorkbench** را انتخاب کنید و مراحل نصب را به اتمام برسانید.

4. اپلیکیشن را اجرا کرده و روی **MySQLConnection** کلیک کنید تا رشته اتصال جدیدی تعریف کنید. رشته اتصالی که در مراحل

قبل از Azure MySQL Database کپی کردید را اینجا استفاده کنید. بعنوان مثال:

Connection Name

: AzureDB;

Host Name

: us-cdb-azure-west-b.cleardb.com;

Username

: <username>;

Password

: <password>;

Default Schema

: IdentityMySQLDatabase

5. پس از برقراری ارتباط با دیتابیس، یک برگ **Query** جدید باز کنید. فرامین زیر را برای ایجاد جداول مورد نیاز کپی کنید.

```
CREATE TABLE `IdentityMySQLDatabase`.`users` (
  `Id` VARCHAR(45) NOT NULL,
  `UserName` VARCHAR(45) NULL,
  `PasswordHash` VARCHAR(100) NULL,
  `SecurityStamp` VARCHAR(45) NULL,
  PRIMARY KEY (`id`));

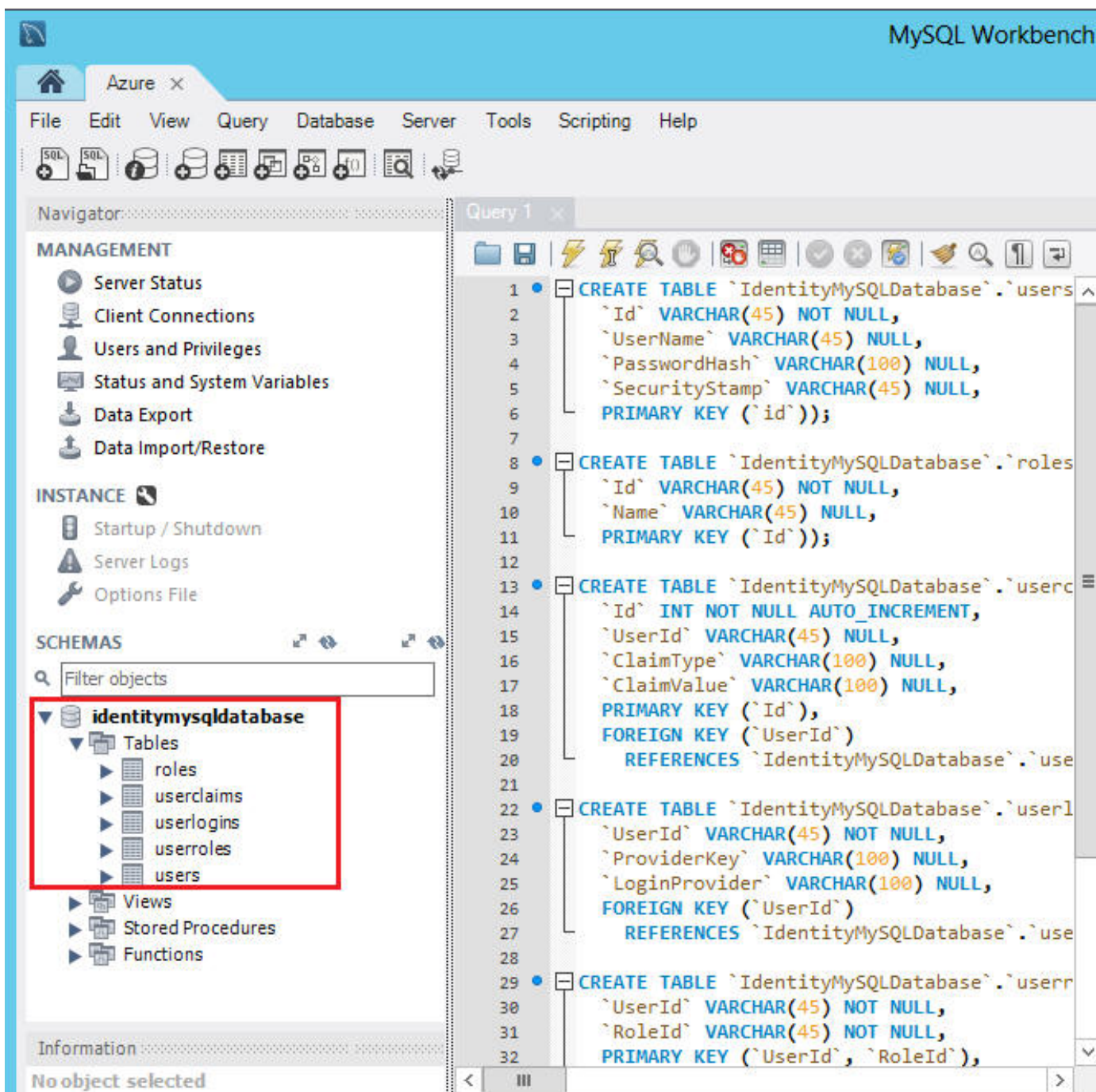
CREATE TABLE `IdentityMySQLDatabase`.`roles` (
  `Id` VARCHAR(45) NOT NULL,
  `Name` VARCHAR(45) NULL,
  PRIMARY KEY (`Id`));

CREATE TABLE `IdentityMySQLDatabase`.`userclaims` (
  `Id` INT NOT NULL AUTO_INCREMENT,
  `UserId` VARCHAR(45) NULL,
  `ClaimType` VARCHAR(100) NULL,
  `ClaimValue` VARCHAR(100) NULL,
  PRIMARY KEY (`Id`),
  FOREIGN KEY (`UserId`)
    REFERENCES `IdentityMySQLDatabase`.`users` (`Id`) on delete cascade);

CREATE TABLE `IdentityMySQLDatabase`.`userlogins` (
  `UserId` VARCHAR(45) NOT NULL,
  `ProviderKey` VARCHAR(100) NULL,
  `LoginProvider` VARCHAR(100) NULL,
  FOREIGN KEY (`UserId`)
    REFERENCES `IdentityMySQLDatabase`.`users` (`Id`) on delete cascade);

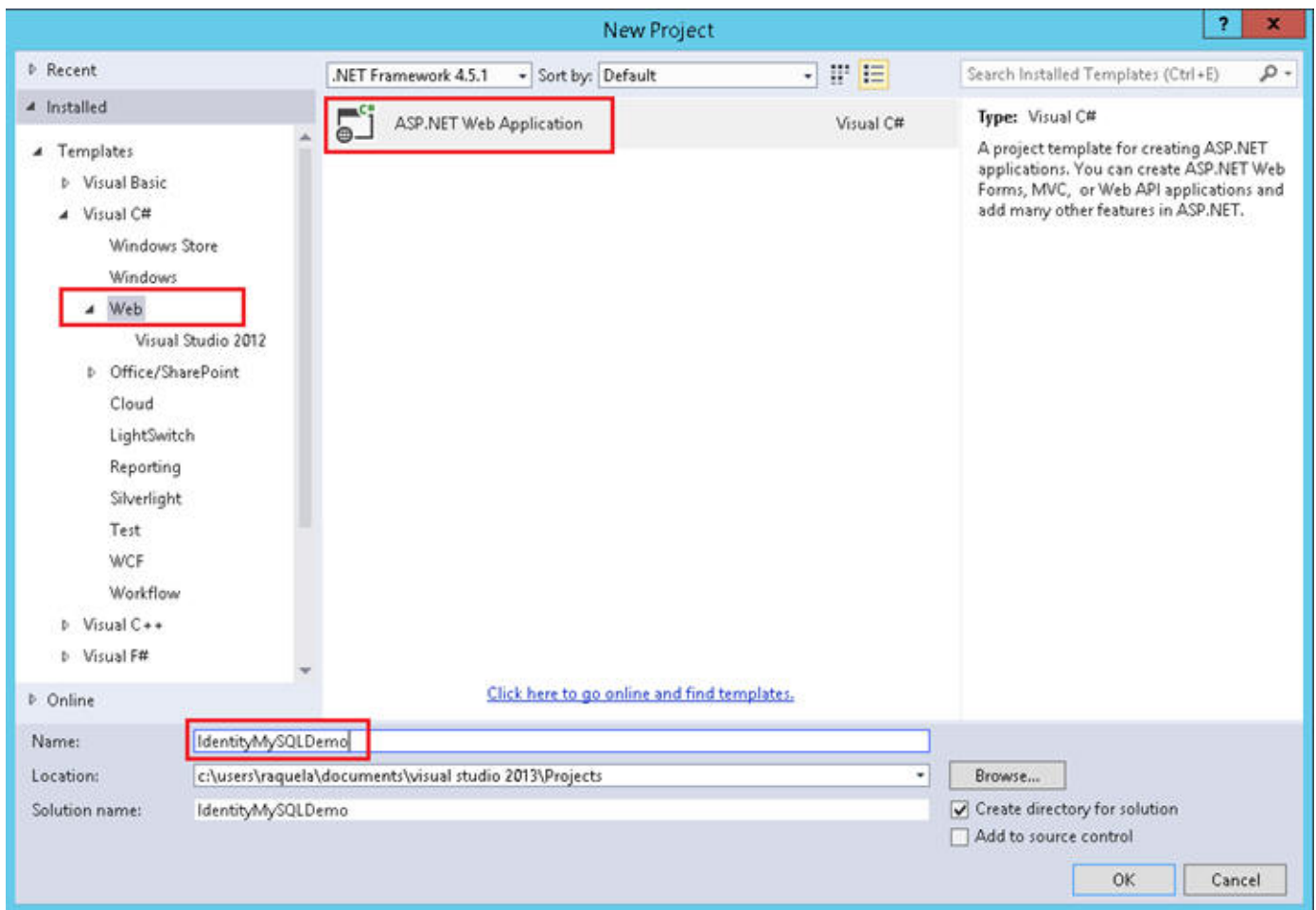
CREATE TABLE `IdentityMySQLDatabase`.`userroles` (
  `UserId` VARCHAR(45) NOT NULL,
  `RoleId` VARCHAR(45) NOT NULL,
  PRIMARY KEY (`UserId`, `RoleId`),
  FOREIGN KEY (`UserId`)
    REFERENCES `IdentityMySQLDatabase`.`users` (`Id`)
    on delete cascade
    on update cascade,
  FOREIGN KEY (`RoleId`)
    REFERENCES `IdentityMySQLDatabase`.`roles` (`Id`)
    on delete cascade
    on update cascade);
```

6. حالا تمام جداول لازم برای ASP.NET Identity را در اختیار دارید، دیتابیس ما MySQL است و روی Windows Azure میزبانی شده.

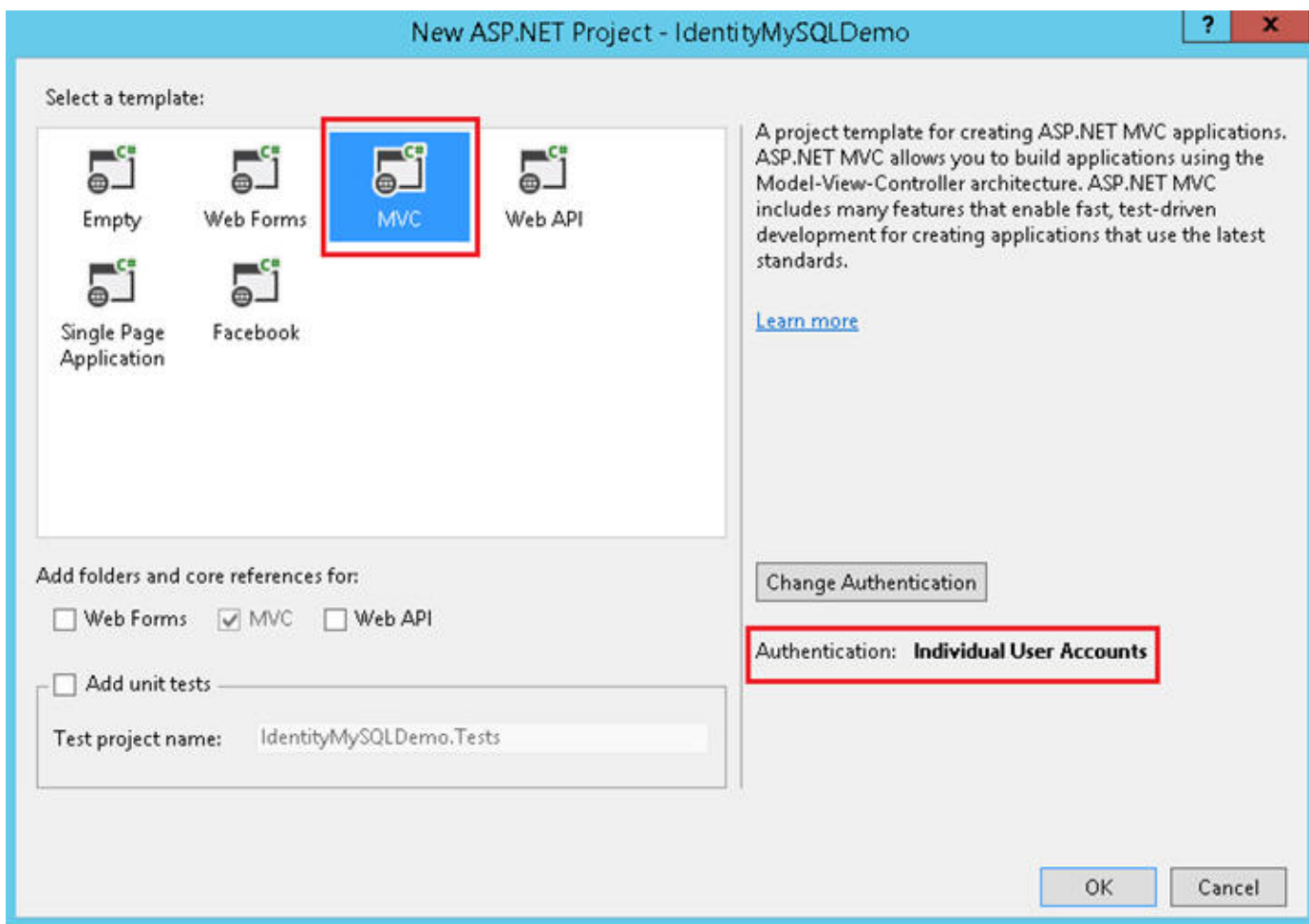


ایجاد یک اپلیکیشن ASP.NET MVC و پیگر بندی آن برای استفاده از MySQL Provider

1. به مخزن <https://github.com/raquelsa/AspNet.Identity.MySQL> بروید.
2. در گوشه سمت راست پایین صفحه روی دکمه Download Zip کلیک کنید تا کل پروژه را دریافت کنید.
3. محتوای فایل دریافتی را در یک پوشه محلی استخراج کنید.
4. پروژه AspNet.Identity.MySQL را باز کرده و آن را کامپایل (build) کنید.
5. روی نام پروژه کلیک راست کنید و گزینه Add, New Project را انتخاب نمایید. پروژه جدیدی از نوع ASP.NET Web Application بسازید و نام آن را به IdentityMySQLDemo تغییر دهید.



6. در پنجره New ASP.NET Project قالب MVC را انتخاب کنید و تنظیمات پیش فرض را بپذیرید.



7. در پنجره Solution Explorer روی پروژه IdentityMySQLDemo کلیک راست کرده و **Manage NuGet Packages** را انتخاب کنید. در قسمت جستجوی دیالوگ باز شده عبارت "Identity.EntityFramework" را وارد کنید. در لیست نتایج این پکیج را انتخاب کرده و آن را حذف (Uninstall) کنید. پیغامی مبنی بر حذف وابستگی‌ها باید دریافت کنید که مربوط به پکیج EntityFramework است، گزینه Yes را انتخاب کنید. از آنجا که کاری با پیاده سازی فرض نخواهیم داشت، این پکیج‌ها را حذف می‌کنیم.

8. روی پروژه IdentityMySQLDemo کلیک راست کرده و **Add, Reference, Solution, Projects** را انتخاب کنید. در دیالوگ باز شده پروژه **AspNet.Identity.MySQL** را انتخاب کرده و OK کنید.

9. در پروژه IdentityMySQLDemo پوشه Models را پیدا کرده و کلاس **IdentityModels.cs** را حذف کنید.

10. در پروژه IdentityMySQLDemo تمام ارجاعات "using Microsoft.AspNet.Identity.EntityFramework;" را با "using;" جایگزین کنید.

11. در پروژه IdentityMySQLDemo تمام ارجاعات به کلاس "ApplicationUser" را با "IdentityUser" جایگزین کنید.

12. کنترلر Account را باز کنید و متد سازنده آنرا مطابق لیست زیر تغییر دهید.

```
public AccountController() : this(new UserManager<IdentityUser>(new UserStore(new MySQLDatabase()))){
}
```

13. فایل web.config را باز کنید و رشته اتصال DefaultConnection را مطابق لیست زیر تغییر دهید.

```
<add name="DefaultConnection" connectionString="Database=IdentityMySQLDatabase;Data Source=<DataSource>;User Id=<UserID>;Password=<Password>" providerName="MySql.Data.MySqlClient" />
```

مقادیر <UserId>, <DataSource> و <Password> را با اطلاعات دیتابیس خود جایگزین کنید.

اجرای اپلیکیشن و اتصال به دیتابیس MySQL

1. روی پروژه IdentityMySQLDemo کلیک راست کرده و **Set as Startup Project** را انتخاب کنید.
2. اپلیکیشن را با Ctrl + F5 کامپایل و اجرا کنید.
3. در بالای صفحه روی **Register** کلیک کنید.
4. حساب کاربری جدیدی بسازید.

[Application name](#) [Home](#) [About](#) [Contact](#)

Register.

Create a new account.

User name

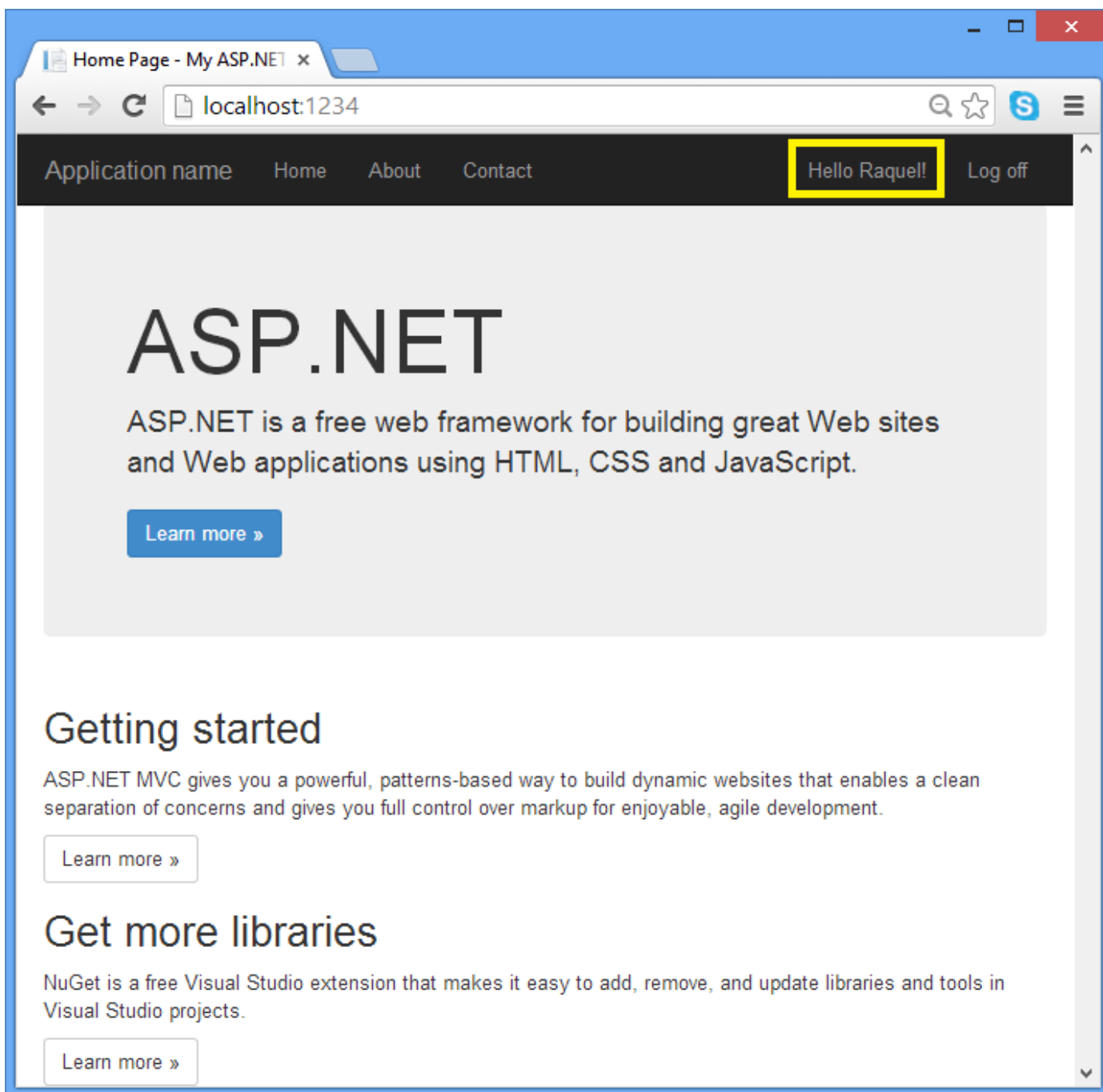
Password

Confirm password

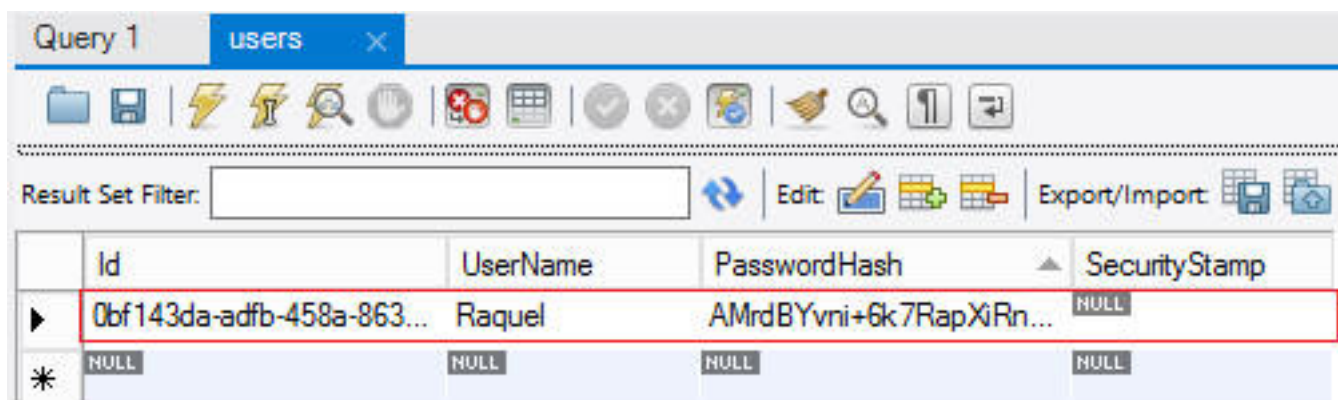
Register

© 2013 - My ASP.NET Application

5. در این مرحله کاربر جدید باید ایجاد شده و وارد سایت شود.



6. به ابزار MySQL Workbench بروید و محتوای جداول **IdentityMySQLDatabase** را بررسی کنید. جدول **users** را باز کنید و اطلاعات کاربر جدید را بررسی نمایید.



	Id	UserName	PasswordHash	SecurityStamp
►	0bf143da-adfb-458a-863...	Raquel	AMrdBYvni+6k7RapXiRn...	NULL
*	NULL	NULL	NULL	NULL

برای ساده نگاه داشتن این مقاله از بررسی تمام کدهای لازم خودداری شده، اما اگر مراحل را دنبال کنید و سورس کد نمونه را دریافت و بررسی کنید خواهید دید که پیاده سازی تامین کنندگان سفارشی برای ASP.NET Identity کار نسبتاً ساده ای است.