بسم الله الرحمن الرحیم

آموزشگاه تحلیل داده

تخصصی ترین مرکز برنامه نویسی و دیتابیس در ایران

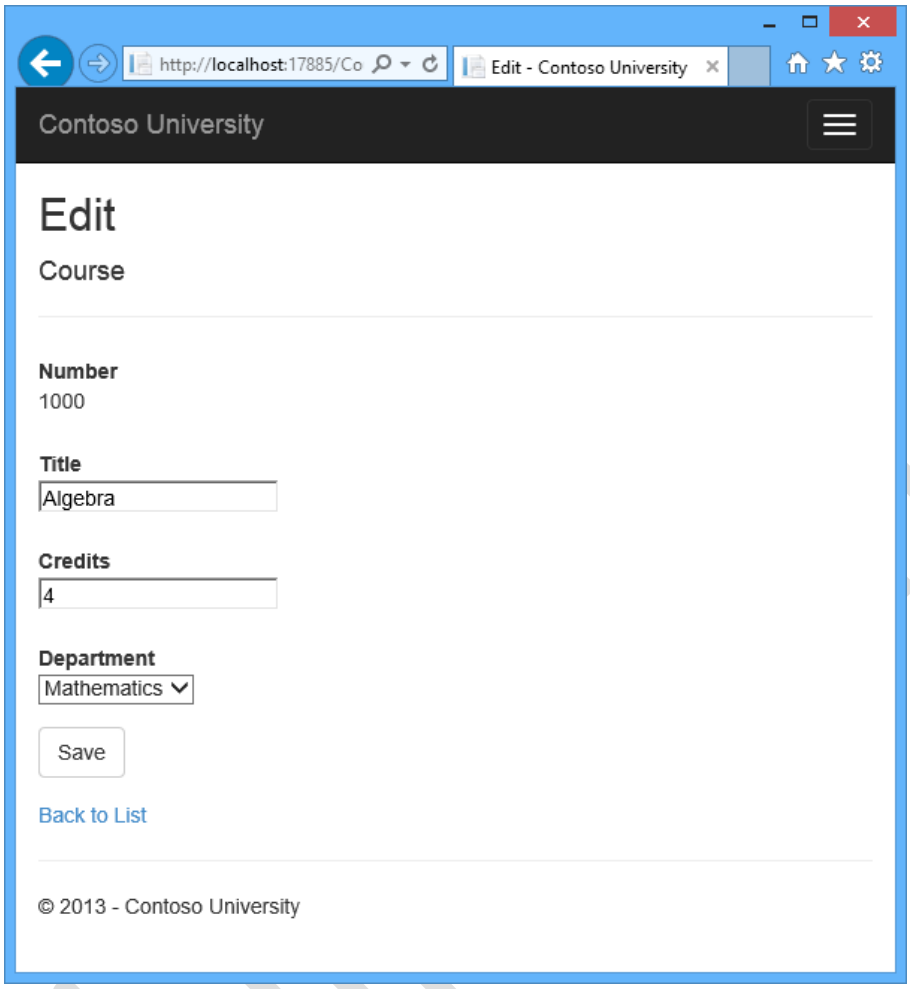
بروز رسانی داده های مربوطه با تکنولوژی EF در MVC

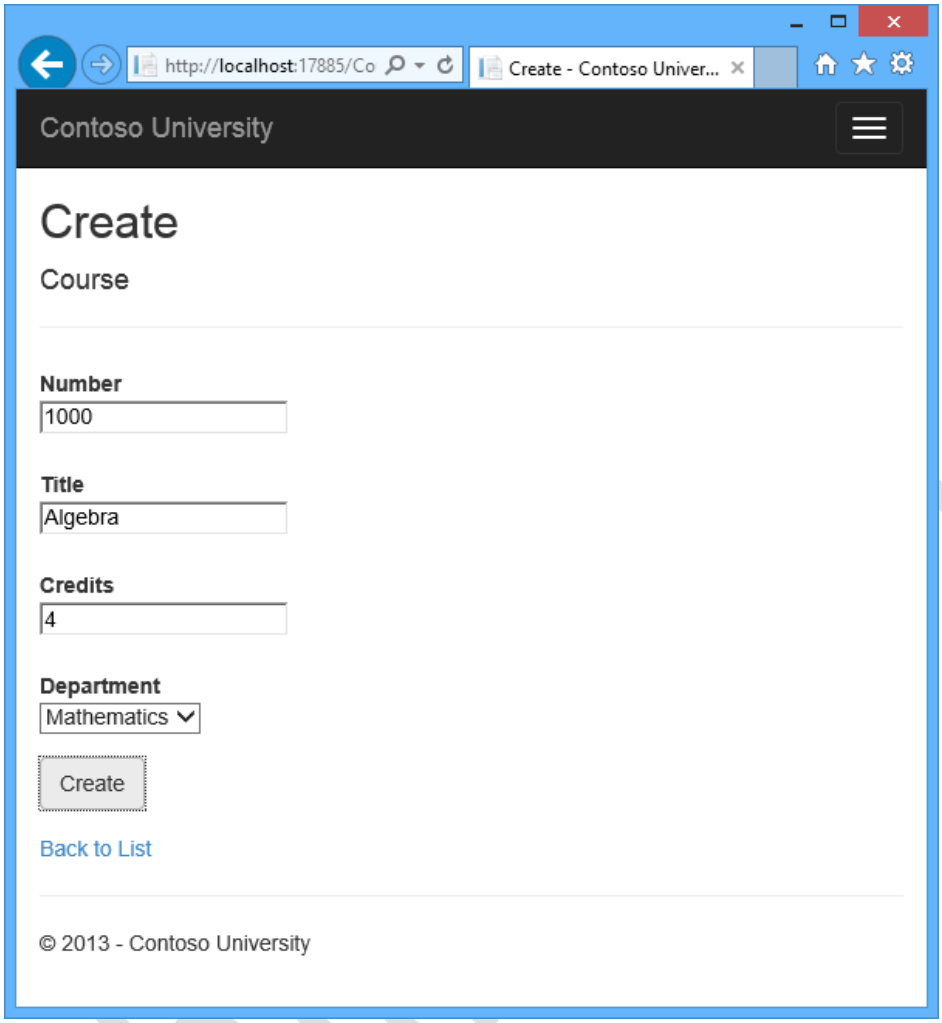
مدرس : مهندس افشین رفوآ

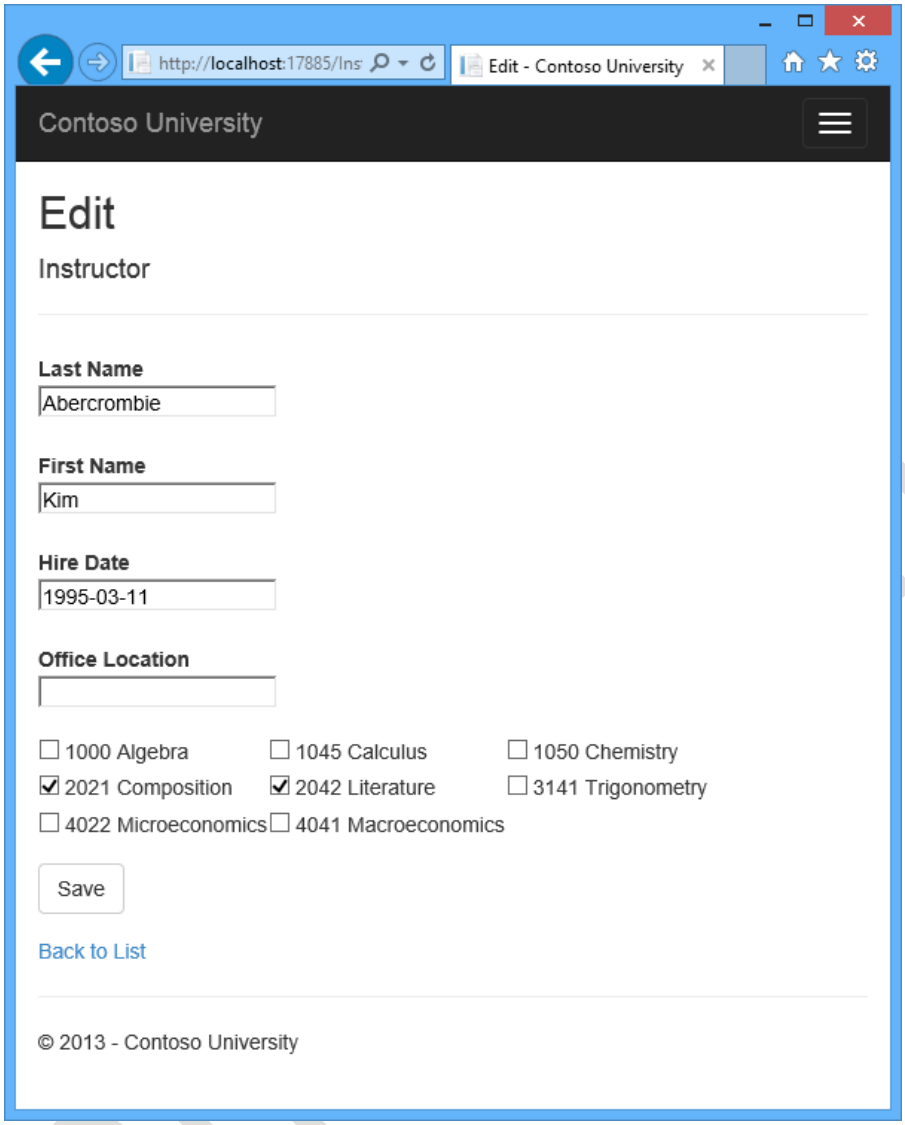
دوره آموزش MVC

بروز رسانی داده های مربوطه با تکنولوژی EF در یک برنامه ی تحت وب ASP.NET MVC

در فصل قبلی داده های مربوطه را نمایش دادیم، حال در مبحث حاضر به روز آوری آن داده ها خواهیم پرداخت. در اکثر رابطه ها، این کار با بروز رسانی foreign key field (فیلد کلید خارجی) و یا navigation property ها امکان پذیر می باشد. در رابطه های چند به چند، EF جدول واسط (join table) را به صورت مستقیم نشان نمی دهد. بنابراین شما موجودیت ها را به navigation property مربوطه اضافه کرده یا از آن حذف می کنید. تصویر زیر برخی از صفحاتی که طی این آموزش با آن ها کار خواهید کرد را نمایش می دهد.


http://localhost:17885/Co
Edit - Contoso University
Contoso University
Edit
Course
Number
1000
Title
Algebra
Credits
4
Department
Mathematics
Save
Back to List
© 2013 - Contoso University


http://localhost:17885/Co
Create - Contoso Univer...
Contoso University
Create
Course
Number
1000
Title
Algebra
Credits
4
Department
Mathematics
Create
Back to List
© 2013 - Contoso University



تنظیم سفارشی صفحات Create و Edit موجودیت Course

پس از اینکه یک موجودیت جدید course ایجاد می‌شود، آن موجودیت بایستی رابطه ای با department موجود داشته باشد. به منظور ایجاد سهولت بیشتر در انجام این کار، کدی که توسط پروسه ی Scaffolding ایجاد می‌شود، شامل متدهای controller و همچنین view های Edit و Create می باشد. View های نام برده خود یک لیست کشویی یا drop-down list را شامل می شود که جهت انتخاب department دلخواه کاربرد دارد. Drop-down list خاصیت کلید خارجی Course.DepartmentID را تنظیم می کند. این تمام آن چیزی است که EF برای بارگذاری خاصیت پیمایشی (navigation property) Department به همراه

موجودیت مربوطه ی Department به آن نیاز دارد. کد ایجاد شده توسط scaffolding را بکار برده، اما به منظور افزودن قابلیت مدیریت خطا و مرتب سازی drop-down list آن را کمی تغییر می دهیم.

فایل CourseController.cs را باز کرده چهار متد Edit و Create را حذف کنید و کد زیر را جایگزین آن ها نمایید:

```
public ActionResult Create()
{
  PopulateDepartmentsDropDownList();
  return View();
}

[HttpPost]
[ValidateAntiForgeryToken]
public ActionResult Create([Bind(Include = "CourseID,Title,Credits,DepartmentID")]Course course)
{
  try
  {
    if (ModelState.IsValid)
    {
      db.Courses.Add(course);
      db.SaveChanges();
      return RedirectToAction("Index");
    }
  }
  catch (RetryLimitExceededException /* dex */)
  {
    //Log the error (uncomment dex variable name and add a line here to write a log.)
    ModelState.AddModelError("", "Unable to save changes. Try again, and if the problem persists, see your system
administrator.");
  }
  PopulateDepartmentsDropDownList(course.DepartmentID);
  return View(course);
}

public ActionResult Edit(int? id)
{
  if (id == null)
  {
    return new HttpStatusCodeResult(HttpStatusCode.BadRequest);
  }
  Course course = db.Courses.Find(id);
  if (course == null)
  {
    return HttpNotFound();
  }
  PopulateDepartmentsDropDownList(course.DepartmentID);
```

```
    return View(course);
}

[HttpPost, ActionName("Edit")]
[ValidateAntiForgeryToken]
public ActionResult EditPost(int? id)
{
    if (id == null)
    {
        return new HttpStatusCodeResult(HttpStatusCode.BadRequest);
    }
    var courseToUpdate = db.Courses.Find(id);
    if (TryUpdateModel(courseToUpdate, "",
       new string[] { "Title", "Credits", "DepartmentID" }))
    {
        try
        {
            db.SaveChanges();

            return RedirectToAction("Index");
        }
        catch (RetryLimitExceededException /* dex */)
        {
            //Log the error (uncomment dex variable name and add a line here to write a log.
            ModelState.AddModelError("", "Unable to save changes. Try again, and if the problem persists, see your system
administrator.");
        }
    }
    PopulateDepartmentsDropDownList(courseToUpdate.DepartmentID);
    return View(courseToUpdate);
}

private void PopulateDepartmentsDropDownList(object selectedDepartment = null)
{
    var departmentsQuery = from d in db.Departments
                           orderby d.Name
                           select d;
    ViewBag.DepartmentID = new SelectList(departmentsQuery, "DepartmentID", "Name", selectedDepartment);
}
```

دستور Using را به ابتدای فایل اضافه کنید:

```
using System.Data.Entity.Infrastructure;
```

متد PopulateDepartmentsDropDownList لیستی از تمامی department ها بازیابی می کند که بر اساس اسم مرتب شده، همچنین یک مجموعه SelectList برای drop-down list ایجاد می کند و بعد مجموعه ی مورد نظر را به view در خاصیت ViewBag ارسال می کند. متد ذکر شده selectedDepartment را به

عنوان پارامتر اختیاری می پذیرد. این پارامتر به کد فراخواننده اجازه می دهد آن آیتمی که به هنگام اجرا و نمایان شدن لیست کشویی انتخاب می شود را تعیین کند. view مورد نظر اسم DepartmentID را به DropDownList helper پاس می دهد. پس از آن helper می داند که برای یافتن SelectList ای که نام آن DepartmentID است، باید درون ViewBag جستجو کند.

متد HttpGet Create، متد PopulateDepartmentsDropDownList را بدون تنظیم آیتم انتخابی صدا می زند، زیرا دپارتمان (department) دوره ی آموزشی (course) جدید هنوز ایجاد نشده:

```
public ActionResult Create()
{
  PopulateDepartmentsDropDownList();
  return View();
}
```

متد HttpGet Edit آیتم انتخابی را بر اساس ID آن department ای که از پیش به course در حال ویرایش اختصاص داده شده است، تنظیم می کند:

```
public ActionResult Edit(int? id)
{
  if (id == null)
  {
    return new HttpStatusCodeResult(HttpStatusCode.BadRequest);
  }
  Course course = db.Courses.Find(id);
  if (course == null)
  {
    return HttpNotFound();
  }
  PopulateDepartmentsDropDownList(course.DepartmentID);
  return View(course);
}
```

متدهای HttpPost، برای Create و Edit کدهایی را شامل می شود که آیتم انتخابی را هنگام مواجه با خطا، یعنی پس از نمایش مجدد صفحه، تنظیم می کند:

```
 catch (RetryLimitExceededException /* dex */)
 {
   //Log the error (uncomment dex variable name and add a line here to write a log.)
   ModelState.AddModelError("", "Unable to save changes. Try again, and if the problem persists, see your system
administrator.");
 }
 PopulateDepartmentsDropDownList(course.DepartmentID);
 return View(course);
```

این کد باعث می شود زمانی که صفحه برای نشان دادن پیام خطا مجدد بارگذاری (نمایش داده) می شود، هر department ای که قبلا انتخاب شده بود، مجددا نمایش داده شود.

View های Course که از پیش توسط scaffolding ایجاد شده، حاوی drop down list هایی برای فیلد department هستند، اما نمی خواهید عنوان (caption) برای این فیلد نمایش داده شود. جهت ویرایش عنوان فیلد، تغییرات هایلایت شده ی زیر را به فایل Views\Course\Create.cshtml اعمال کنید.

```
@model ContosoUniversity.Models.Course

@{
  ViewBag.Title = "Create";
}

<h2>Create</h2>


@using (Html.BeginForm())
{
  @Html.AntiForgeryToken()

<div class="form-horizontal">
  <h4>Course</h4>
  <hr />
  @Html.ValidationSummary(true)

  <div class="form-group">
    @Html.LabelFor(model => model.CourseID, new { @class = "control-label col-md-2" })
    <div class="col-md-10">
      @Html.EditorFor(model => model.CourseID)
      @Html.ValidationMessageFor(model => model.CourseID)
    </div>
  </div>

  <div class="form-group">
    @Html.LabelFor(model => model.Title, new { @class = "control-label col-md-2" })
    <div class="col-md-10">
      @Html.EditorFor(model => model.Title)
      @Html.ValidationMessageFor(model => model.Title)
    </div>
  </div>

  <div class="form-group">
    @Html.LabelFor(model => model.Credits, new { @class = "control-label col-md-2" })
    <div class="col-md-10">
      @Html.EditorFor(model => model.Credits)
      @Html.ValidationMessageFor(model => model.Credits)
    </div>
```

```
    </div>

    <div class="form-group">
      <label class="control-label col-md-2" for="DepartmentID">Department</label>
      <div class="col-md-10">
        @Html.DropDownList("DepartmentID", String.Empty)
        @Html.ValidationMessageFor(model => model.DepartmentID)
      </div>
    </div>

    <div class="form-group">
      <div class="col-md-offset-2 col-md-10">
        <input type="submit" value="Create" class="btn btn-default" />
      </div>
    </div>
  </div>
}

<div>
  @Html.ActionLink("Back to List", "Index")
</div>

@section Scripts {
  @Scripts.Render("~/bundles/jqueryval")
}
```

حال همین تغییرات را در فایل Views\Course\Edit.cshtml نیز پیاده کنید.

به طور معمول scaffolder کلید اصلی (primary key) را ایجاد نمی کند، زیرا مقدار کلید اصلی توسط پایگاه داده ایجاد شده و غیرقابل تغییر می باشد و از آنجایی که این مقدار معنی دار و قابل فهم نیست آن را نمی توان برای کاربر نمایش داد. Scaffolder برای فیلد CourseID یک کادر متن (textbox) نیز ایجاد می کند، زیرا برای آن تعریف شده (می داند) که هر زمان که خصیصه ی DatabaseGeneratedOption.None بکار رفت معنیش این است که کاربر باید بتواند مقدار کلید اصلی را وارد کند، اما برای آن تعریف نشده که به دلیل معنی دار بودن عدد می خواهید آن عدد در view های دیگر نمایش داده شود. به همین خاطر بایستی آن را خود به صورت دستی اضافه کنید.

در فایل Views\Course\Edit.cshtml یک فیلد به نام Number قبل از فیلد Title اضافه کنید. به این خاطر که یک primary key است، به نمایش گذاشته می شود ولی امکان تغییر آن وجود ندارد.

```
<div class="form-group">
```

```
    @Html.LabelFor(model => model.CourseID, new { @class = "control-label col-md-2" })
    <div class="col-md-10">
        @Html.DisplayFor(model => model.CourseID)
    </div>
</div>
```

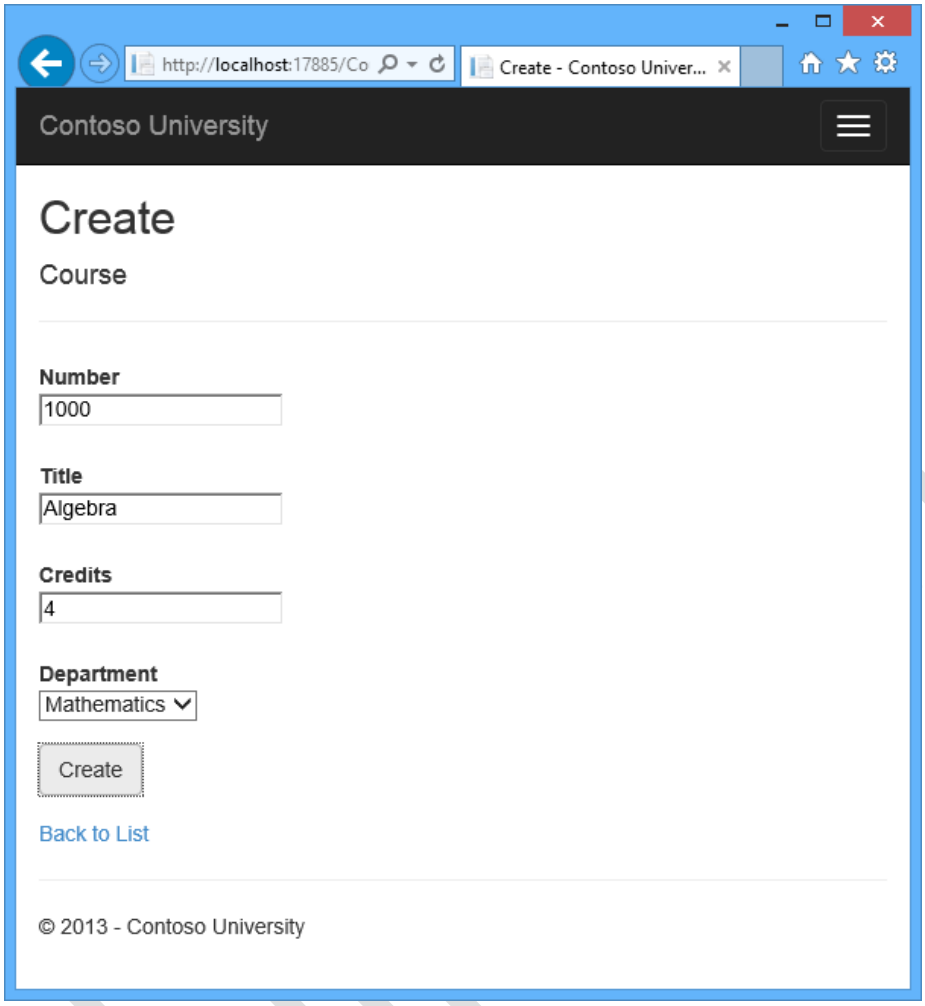
یک فیلد مخفی (Html.HiddenFor helper) از قبل برای course number در Edit view ایجاد شده.

افزودن Html.LabelFor helper، نیاز به فیلد مخفی را از میان بر نمی دارد. دلیلش این است که استفاده از Html.LabelFor باعث نمی شود course number در اطلاعات ارسالی، هنگامی که کاربر دکمه ی Save را بر روی صفحه ی Edit کلیک می کند، گنجانده شود.

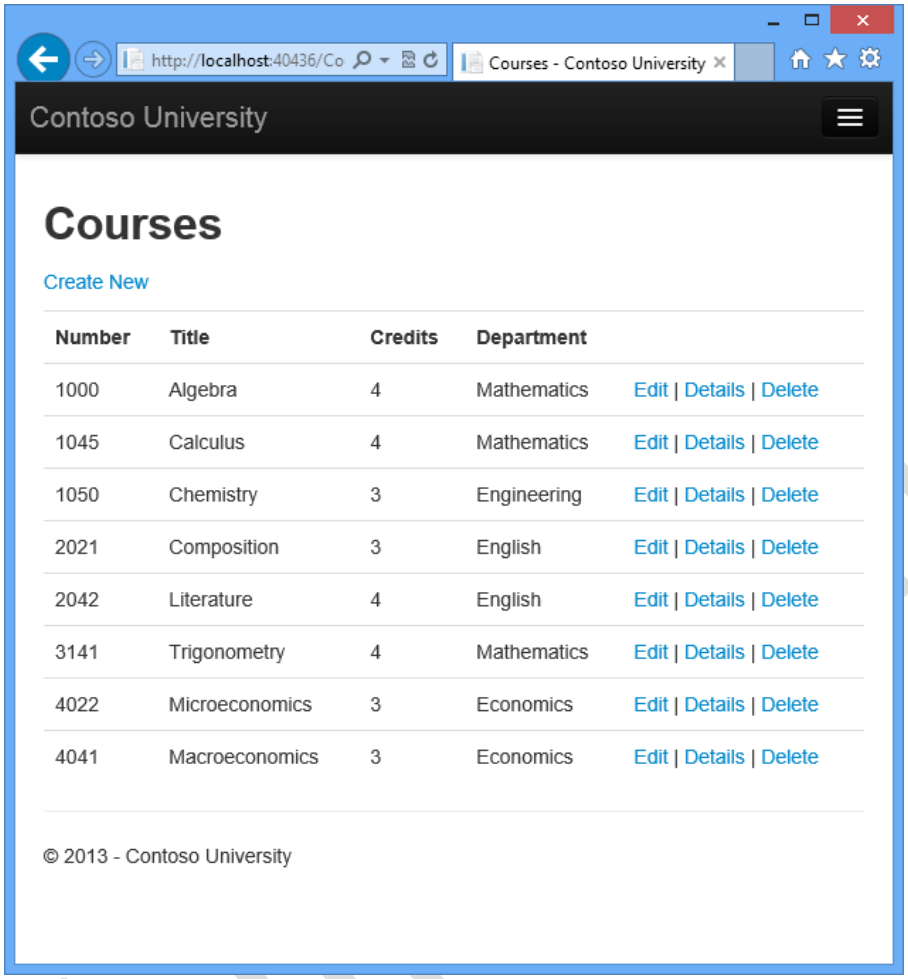
در فایل های Views\Course\Delete.cshtml و Views\Course\Details.cshtml، عنوان department را از "Name" به "Department" تغییر داده و یک فیلد به نام Course پیش از فیلد Title اضافه کنید.

```
<dt>
    Department
</dt>

<dd>
    @Html.DisplayFor(model => model.Department.Name)
</dd>

<dt>
    @Html.DisplayNameFor(model => model.CourseID)
</dt>

<dd>
    @Html.DisplayFor(model => model.CourseID)
</dd>
```

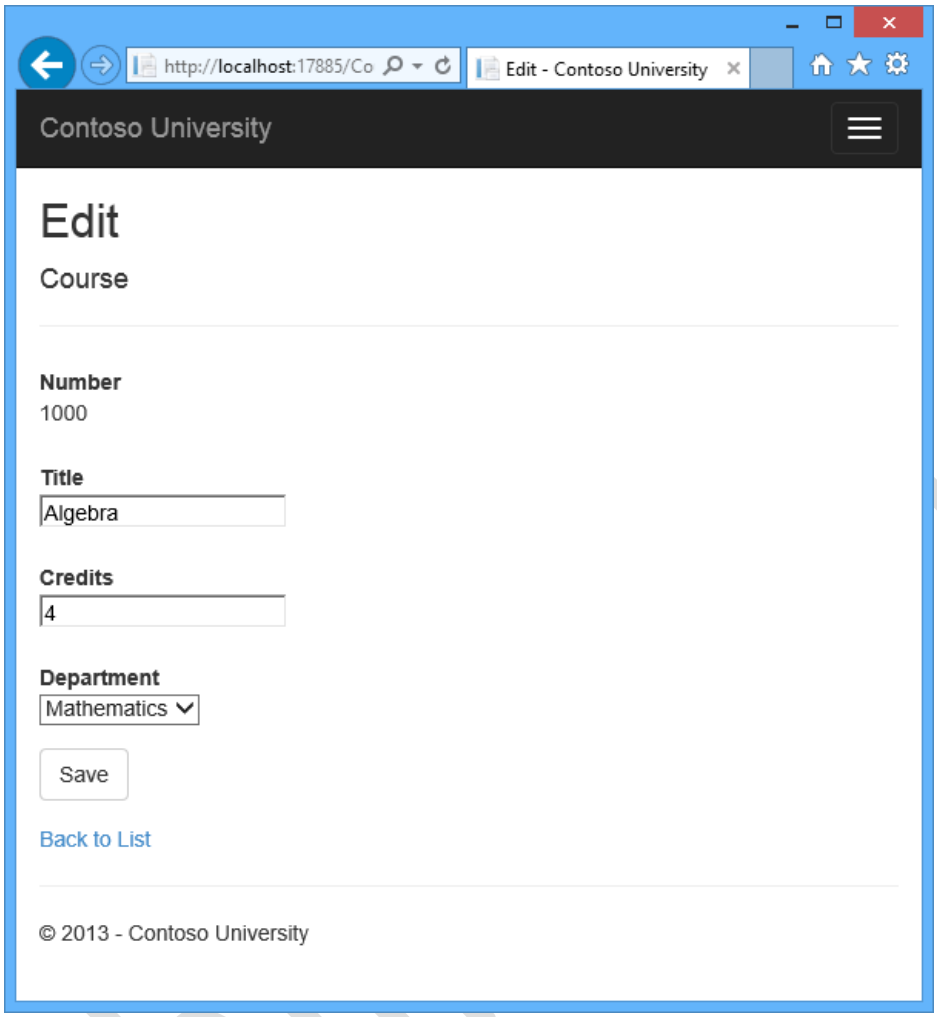
صفحه ی Create را اجرا کرده (صفحه ی Course Index را نمایش داده و Create New را کلیک کنید)، سپس داده های course جدید را وارد نمایید:



Create را کلیک کنید. صفحه ی Course Index در حالی نمایش داده می شود که course جدید به آن اضافه شده است. اسم department داخل لیست صفحه ی Index در واقع از یک navigation property گرفته می شود که نشانگر درست برقرار شدن رابط می باشد.



صفحه ی Edit را اجرا کنید (صفحه ی Course Index را اجرا کرده و بر روی لینک Edit برای course مشخص کلیک کنید).



اطلاعات مورد نظر را ویرایش کرده و Save را کلیک کنید. صفحه ی Course Index به همراه اطلاعات و داده های روز آوری شده ی course نمایش داده می شود.

ایجاد یک صفحه ی Edit ویژه ی Instructor ها

پس از اینکه رکورد یا سطر مربوط به یک instructor را ویرایش کردیم، می خواهیم office assignment آن instructor را نیز بروز آوری کنیم. موجودیت Instructor دارای یک رابطه ی یک به صفر یا یک به یک با موجودیت OfficeAssignment می باشد، بدین معنا که بایستی سناریوهای زیر را در نظرگرفته و مدیریت کنید:

1. چنانچه office assignment دارای مقداری بود و کاربر آن را پاک کرد، در آن صورت باید موجودیت OfficeAssignment را به طور کلی حذف نمود.

2. اگر office assignment تهی بود و کاربر مقداری را در آن وارد کرد، در آن صورت می بایست یک موجودیت OfficeAssignment جدید ایجاد نمود.

3. اگر کاربر مقدار یک office assignment را تغییر داد، بایستی آن مقدار را در یک موجودیت OfficeAssignment که از قبل موجود می باشد، ویرایش کرد.

فایل InstructorController.cs را باز کرده و متد HttpGet Edit را بررسی کنید:

```
{
    if (id == null)
    {
      return new HttpStatusCodeResult(HttpStatusCode.BadRequest);
    }
    Instructor instructor = db.Instructors.Find(id);
    if (instructor == null)
    {
      return HttpNotFound();
    }
    ViewBag.ID = new SelectList(db.OfficeAssignments, "InstructorID", "Location", instructor.ID);
    return View(instructor);
 }
```

کدی که توسط scaffolding ایجاد شده، در اینجا آن چیزی نیست که به آن نیاز دارید. این کد درواقع یک drop-down list تنظیم می کند، اما آنچه مد نظر شما است یک textbox می باشد. کد زیر را جایگزین این متد کنید:

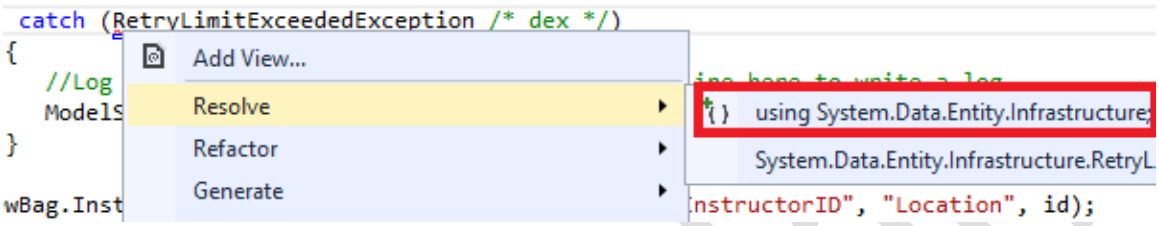
```
  public ActionResult Edit(int? id)
{
  if (id == null)
  {
    return new HttpStatusCodeResult(HttpStatusCode.BadRequest);
  }
  Instructor instructor = db.Instructors
    .Include(i => i.OfficeAssignment)
    .Where(i => i.ID == id)
    .Single();
  if (instructor == null)
  {
    return HttpNotFound();
  }
  return View(instructor);
}
```

این کد دستور ViewBag را حذف می کند و قابلیت eager loading را برای موجودیت OfficeAssignment مربوطه اضافه می نماید. امکان اجرای eager loading با متد Find وجود ندارد، به همین خاطر جهت انتخاب instructor از متدهای Where و Single به عنوان جایگزین استفاده می شود.

متد HttpPost Edit را با کد زیر جایگزین کنید. این کد بروز رسانی های office assignment را مدیریت می کند:

```
[HttpPost, ActionName("Edit")]
[ValidateAntiForgeryToken]
public ActionResult EditPost(int? id)
{
  if (id == null)
  {
    return new HttpStatusCodeResult(HttpStatusCode.BadRequest);
  }
  var instructorToUpdate = db.Instructors
    .Include(i => i.OfficeAssignment)
    .Where(i => i.ID == id)
    .Single();

  if (TryUpdateModel(instructorToUpdate, "",
    new string[] { "LastName", "FirstMidName", "HireDate", "OfficeAssignment" }))
  {
    try
    {
      if (String.IsNullOrWhiteSpace(instructorToUpdate.OfficeAssignment.Location))
      {
        instructorToUpdate.OfficeAssignment = null;
      }

      db.SaveChanges();

      return RedirectToAction("Index");
    }
    catch (RetryLimitExceededException /* dex */)
    {
      //Log the error (uncomment dex variable name and add a line here to write a log.
      ModelState.AddModelError("", "Unable to save changes. Try again, and if the problem persists, see your system
administrator.");
    }
  }
  return View(instructorToUpdate);
}
```

ارجاع به RetryLimitExceededException نیازمند افزودن یک دستور using می باشد، به منظور افزودن آن روی RetryLimitExceededException راست کلیک کرده، سپس Resolve - using System.Data.Entity.Infrastructure را کلیک کنید.



این کد کارهای زیر را انجام می دهد:

1. کد حاضر اسم متد مورد نظر را به EditPost تغییر می دهد، زیرا امضای آن (signature) هم اکنون با متد HttpGet یکسان می باشد (خصیصه ی ActionName تعیین می کند که Edit/ URL/ همواره باید بکار گرفته شود).

2. با استفاده از قابلیت eager loading برای خاصیت پیمایشی (navigation property) OfficeAssignment موجودیت جاری Instructor را از پایگاه داده بازیابی می کند. این دقیقا همان کاری است که با متد HttpGet Edit انجام دادید.

3. موجودیت بازیابی شده ی Instructor را با مقادیری از model binder بروز رسانی می کند. نسخه ی overload شده از TryUpdateModel که بکار رفته، به شما این امکان را می دهد که property هایی را که می خواهید اضافه کنید را whitelist (مجاز اعلام) نمایید.

```
if (TryUpdateModel(instructorToUpdate, "",
 new string[] { "LastName", "FirstMidName", "HireDate", "OfficeAssignment" }))
```

4. در صورت خالی بودن office location، خاصیت Instructor.OfficeAssignment را روی null تنظیم می کند تا سطر مربوطه داخل جدول OfficeAssignment حذف گردد.

```
 if (String.IsNullOrWhiteSpace(instructorToUpdate.OfficeAssignment.Location))
{
 instructorToUpdate.OfficeAssignment = null;
```

```
}
```

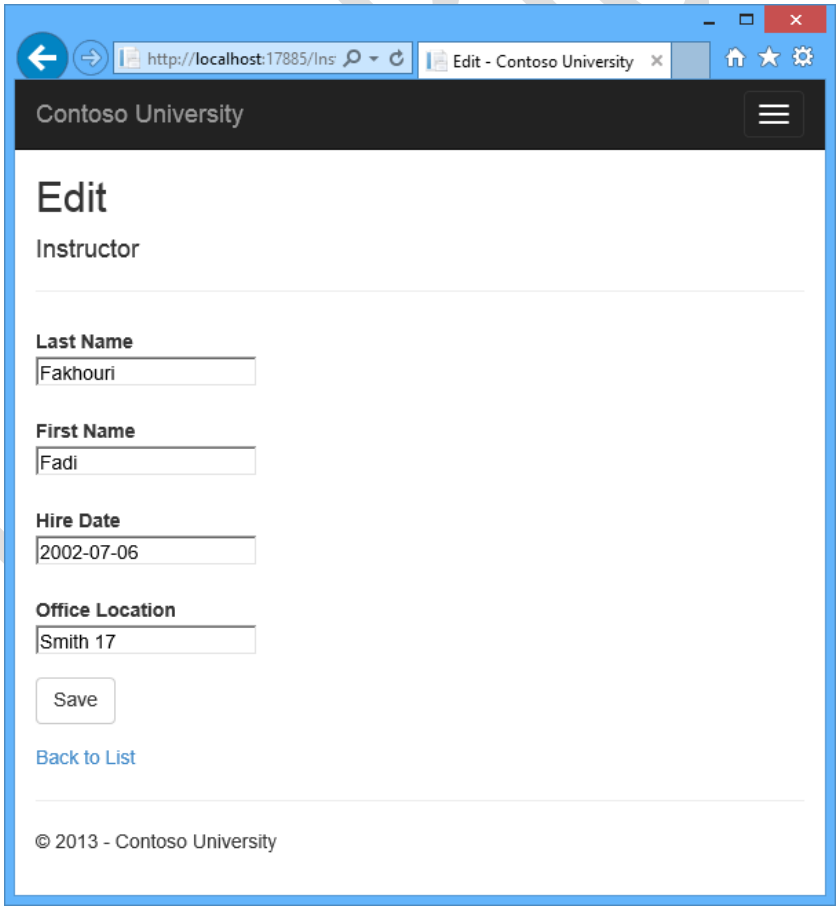
5. در نهایت تغییرات اعمال شده را در پایگاه داده ذخیره می کند.

در فایل Views\Instructor\Edit.cshtml، بعد از المان های div ای که مختص فیلد Hire Date می باشد، یک فیلد جدید برای ویرایش office location اضافه کنید:

```
<div class="form-group">
    @Html.LabelFor(model => model.OfficeAssignment.Location, new { @class = "control-label col-md-2" })
    <div class="col-md-10">
        @Html.EditorFor(model => model.OfficeAssignment.Location)
        @Html.ValidationMessageFor(model => model.OfficeAssignment.Location)
    </div>
</div>
```

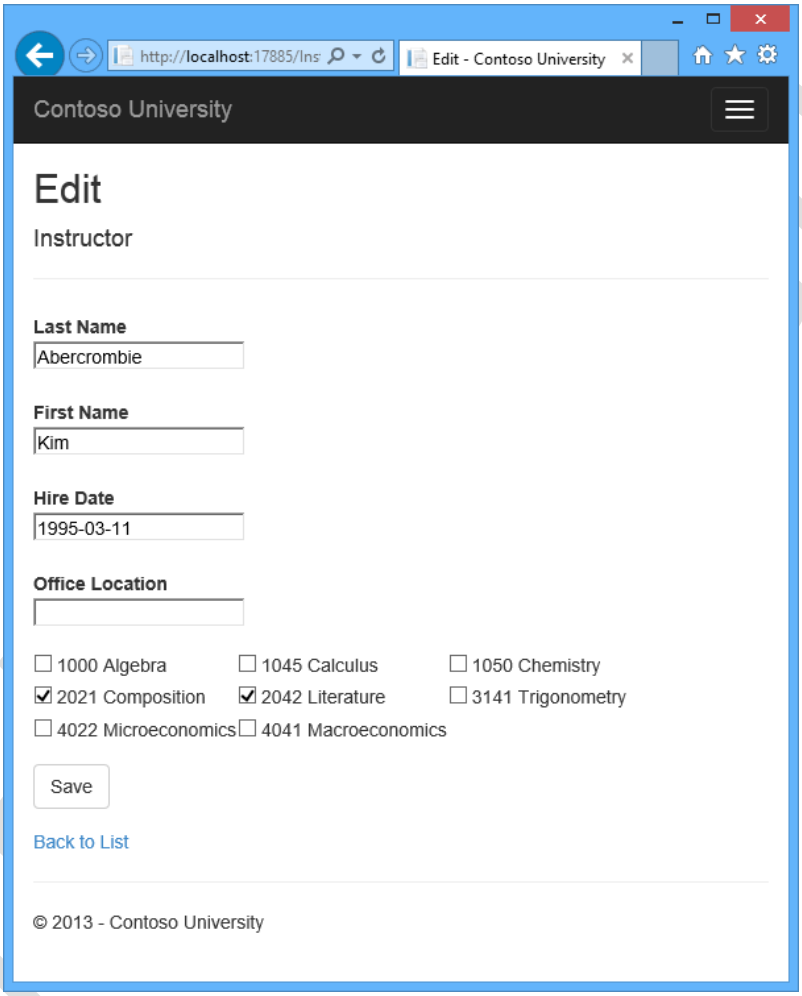
صفحه را اجرا کرده (تب Instructor را انتخاب کرده و روی لینک Edit یکی از Instructor ها کلیک کنید).

Office Location را ویرایش کرده و دکمه ی Save را کلیک کنید.



افزودن Course assignment به صفحه ی Instructor Edit

مدرسان (instructor) می توانند تعداد زیادی دوره ی آموزشی (course) را تدریس کنند. حال با افزودن قابلیت ویرایش course assignment ها (تخصیص دوره به مدرس) با استفاده از یک سری checkbox، این صفحه را ارتقا می دهیم:



رابطه ی بین موجودیت های Course و Instructor از نوع چند به چند می باشد، بدین معنا که شما به خاصیت های کلید خارجی (foreign key property) که در جدول واسط (join table) قرار دارند دسترسی مستقیم ندارید. در عوض، موجودیت ها را از خاصیت پیمایشی Instructor.Courses حذف کرده یا به آن اضافه می کنید.

رابط کاربری (UI) که به شما امکان می دهد تعیین کنید کدام دوره های آموزشی به کدام مدرس تخصیص یابد، یک سری کادر تیک (checkbox) می باشد. به ازای هر course موجود در پایگاه داده، یک کادر تیک نمایش داده می شود و checkbox آن دوره هایی که هم اکنون مدرس به آن ها تخصیص یافته، تیک دار می باشد. کاربر می تواند با تیک دار کردن یا برداشتن تیک checkbox ها، course assignment را ویرایش کند. چنانچه تعداد دوره های آموزشی بسیار بیشتر بود، در آن صورت می بایست یک روشی دیگر را برای ارائه ی اطلاعات در view انتخاب کنید، اما برای ایجاد یا حذف رابطه ها از همان روش های دستکاری و مدیریت navigation property ها بهره گیرید.

جهت ارائه ی اطلاعات و داده ها به view برای checkbox ها، از یک کلاس view model بهره می گیریم. فایل AssignedCourseData.cs را در پوشه ی ViewModels ایجاد کرده و کد زیر را جایگزین کد جاری کنید:

```
namespace ContosoUniversity.ViewModels
{
  public class AssignedCourseData
  {
    public int CourseID { get; set; }
    public string Title { get; set; }
    public bool Assigned { get; set; }
  }
}
```

در فایل InstructorController.cs، کد متد HttpGet Edit را با کد زیر جایگزین کنید. همان طور که مشاهده می کنید، این تغییرات هایلایت شده اند:

```
public ActionResult Edit(int? id)
    {
      if (id == null)
      {
        return new HttpStatusCodeResult(HttpStatusCode.BadRequest);
      }
      Instructor instructor = db.Instructors
        .Include(i => i.OfficeAssignment)
        .Include(i => i.Courses)
        .Where(i => i.ID == id)
        .Single();
      PopulateAssignedCourseData(instructor);
      if (instructor == null)
      {
        return HttpNotFound();
      }
      return View(instructor);
```

```
}

private void PopulateAssignedCourseData(Instructor instructor)
{
  var allCourses = db.Courses;
  var instructorCourses = new HashSet<int>(instructor.Courses.Select(c => c.CourseID));
  var viewModel = new List<AssignedCourseData>();
  foreach (var course in allCourses)
  {
    viewModel.Add(new AssignedCourseData
    {
      CourseID = course.CourseID,
      Title = course.Title,
      Assigned = instructorCourses.Contains(course.CourseID)
    });
  }
  ViewBag.Courses = viewModel;
}
```

این کد eager loading را برای خاصیت پیمایشی Courses تعریف (اضافه) کرده و متد جدید PopulateAssignedCourseData را صدا می زند تا به وسیله ی کلاس AssignedCourseData view model، اطلاعات لازم را برای آرایه ای از checkbox ها فراهم آورد.

کد موجود در متد PopulateAssignedCourseData، تمامی موجودیت های Course را خوانده و فهرستی از course ها را با استفاده از کلاس view model بارگذاری می کند. برای هر course، کد بررسی می کند آیا course مورد نظر در خاصیت پیمایشی Courses آن instructor وجود دارد یا خیر. به منظور ایجاد یک مکانیزم جستجو (lookup) کارآمد که بتوان از آن برای بررسی تخصیص یا عدم تخصیص یک course به instructor استفاده کرد، course های اختصاص یافته به instructor داخل یک مجموعه ی HashSet قرار داده می شوند. خاصیت Assigned، برای course هایی که به instructor تخصیص یافته، بر روی مقدار true تنظیم می شوند. view به کمک این خاصیت تصمیم می گیرد کدام checkbox می بایست به صورت تیک دار نمایش داده شود. سرانجام لیست مورد نظر در قالب یک خاصیت ViewBag به view ارسال می گردد.

اکنون آن کدی را اضافه می کنیم که با کلیک کاربر بر روی دکمه ی Save، اجرا می شود. کد زیر را جایگزین متد EditPost نمایید. این کد سبب فراخوانی متد جدیدی می شود که خاصیت پیمایشی Courses موجودیت Instructor را بروز آوری می کند. این تغییرات با رنگ زرد هایلایت شده:

```
[HttpPost]
[ValidateAntiForgeryToken]
```

```
public ActionResult Edit(int? id, string[] selectedCourses)
{
  if (id == null)
  {
    return new HttpStatusCodeResult(HttpStatusCode.BadRequest);
  }
  var instructorToUpdate = db.Instructors
    .Include(i => i.OfficeAssignment)
    .Include(i => i.Courses)
    .Where(i => i.ID == id)
    .Single();

  if (TryUpdateModel(instructorToUpdate, "",
    new string[] { "LastName", "FirstMidName", "HireDate", "OfficeAssignment" }))
  {
    try
    {
      if (String.IsNullOrWhiteSpace(instructorToUpdate.OfficeAssignment.Location))
      {
        instructorToUpdate.OfficeAssignment = null;
      }

      UpdateInstructorCourses(selectedCourses, instructorToUpdate);

      db.SaveChanges();

      return RedirectToAction("Index");
    }
    catch (RetryLimitExceededException /* dex */)
    {
      //Log the error (uncomment dex variable name and add a line here to write a log.
      ModelState.AddModelError("", "Unable to save changes. Try again, and if the problem persists, see your
system administrator.");
    }
  }
  PopulateAssignedCourseData(instructorToUpdate);
  return View(instructorToUpdate);
}
private void UpdateInstructorCourses(string[] selectedCourses, Instructor instructorToUpdate)
{
  if (selectedCourses == null)
  {
    instructorToUpdate.Courses = new List<Course>();
    return;
  }

  var selectedCoursesHS = new HashSet<string>(selectedCourses);
  var instructorCourses = new HashSet<int>
    (instructorToUpdate.Courses.Select(c => c.CourseID));
  foreach (var course in db.Courses)
```

```
        {
            if (selectedCoursesHS.Contains(course.CourseID.ToString()))
            {
                if (!instructorCourses.Contains(course.CourseID))
                {
                    instructorToUpdate.Courses.Add(course);
                }
            }
            else
            {
                if (instructorCourses.Contains(course.CourseID))
                {
                    instructorToUpdate.Courses.Remove(course);
                }
            }
        }
    }
```

همان طور که مشاهده می کنید، signature (ورودی) متد مورد نظر اکنون دیگر با signature متد HttpGet متفاوت می باشد. از این رو تغییر اسم متد از EditPost به Edit را تجربه می کنیم.

از آنجایی که view دارای مجموعه ای از موجودیت های Course نیست، model binder نمی تواند به صورت خودکار خاصیت پیمایشی Courses را بروز رسانی کند. بجای استفاده از model binder جهت روز آوری خاصیت پیمایشی Courses، این کار را در متد جدید UpdateInstructorCourses انجام دهید. از این رو می بایستی خاصیت Courses را از model binding حذف کنید. برای این کار نیازی به اصلاح کدی که متد TryUpdateModel را صدا می زند نیست، زیرا شما از نسخه های overload شده ی مجاز اعلام شده (whitelist) استفاده می کنید و Course در آن لیست جایی ندارد.

اگر هیچ checkbox ای انتخاب نشده باشد، کد داخل UpdateInstructorCourses خاصیت پیمایشی Courses را با مجموعه ای تهی (empty collection) مقداردهی اولیه (initialize) می کند:

```
if (selectedCourses == null)
{
    instructorToUpdate.Courses = new List<Course>();
    return;
}
```

کد مورد نظر درون course های موجود در پایگاه داده می چرخد (loop) و هر course را با آن course هایی که در حال حاضر به instructor تخصیص یافته چک می کند، سپس course های انتساب داده شده به

instructor را با آن هایی که در view انتخاب شده بود مقایسه می کند. جهت سهولت بخشیدن و بهینه سازی جستجوها، دو مجموعه ی آخری در اشیا HashSet ذخیره می شوند.

اگر checkbox یک course تیک دار یا به عبارتی انتخاب شده باشد در حالی که آن course در خاصیت پیمایشی Instructor.Courses وجود نداشته باشد، در آن صورت course ذکر شده به مجموعه ی مورد نظر در navigation property اضافه می گردد.

```
if (selectedCoursesHS.Contains(course.CourseID.ToString()))
        {
          if (!instructorCourses.Contains(course.CourseID))
          {
            instructorToUpdate.Courses.Add(course);
          }
        }
```

چنانچه checkbox آن course انتخاب نشده باشد، اما course مورد نظر در خاصیت پیمایشی Instructor.Courses وجود داشته باشد، در آن صورت course نام برده از navigation property حذف می شود.

```
else
{
  if (instructorCourses.Contains(course.CourseID))
  {
    instructorToUpdate.Courses.Remove(course);
  }
}
```

فایل Views\Instructor\Edit.cshtml را باز کرده و با افزودن کد زیر بلافاصله پس از المان های div مربوط به فیلد OfficeAssignment و درست قبل از عنصر div دکمه ی Save، یک فیلد به نام Courses به همراه آرایه ای از checkbox ها را به view مورد نظر اضافه کنید:

```
<div class="form-group">
 <div class="col-md-offset-2 col-md-10">
  <table>
   <tr>
    @{
    int cnt = 0;
    List<ContosoUniversity.ViewModels.AssignedCourseData> courses = ViewBag.Courses;

        foreach (var course in courses)
        {
```

```
            if (cnt++ % 3 == 0)
            {
              @:</tr><tr>
    }
    @:<td>
     <input type="checkbox"
       name="selectedCourses"
       value="@course.CourseID"
       @(Html.Raw(course.Assigned=="" ? "checked=\"checked\"" :="" "")) />
     @course.CourseID @: @course.Title
     @:
    </td>
    }
    @:
   </tr>
   }
  </table>
 </div>
</div>
```

پس از جای گذاری کد، چنانچه فاصله ی بین خطوط (line break) و توگذاری ها (indentation) شبیه به آنچه در اینجا مشاهده می کنید نیست، همه چیز را به صورت دستی خودتان تنظیم کنید تا مشابه نمونه ی حاضر شود. ضرورتی ندارد توگذاری صد در صد عالی باشد، اما خط های @</tr><tr>، @:<td>، @:</td> و @</tr> بایستی هر یک بر روی خطی جداگانه نمایش داده شود، در غیر این صورت یک خطای زمان اجرا دریافت می کنید.

این کد یک جدول HTML ایجاد می کند که دارای سه ستون می باشد. در هر ستون یک checkbox و به دنبال آن یک عنوان (caption) که course number و title را شامل می شود، نمایش داده می شود. Check box ها همگی دارای اسم یکسان می باشند ("selectedCourses"). این امر به model binder می فهماند که تمامی checkbox ها می بایست به صورت یک گروه با آن ها برخورد شود. خصیصه ی value هر checkbox بر روی مقدار CourseID تنظیم شده است. هنگامی که صفحه ارسال می گردد، model binder یک آرایه به controller که دربردارنده ی مقادیر CourseID می باشد، ویژه ی آن checkbox هایی که تیک دار هستند(انتخاب شده اند)، ارسال می کند.

زمانی که checkbox ها برای اولین بار نمایش (render) داده می شوند، آن هایی که مربوط به course های تخصیص یافته به instructor هستند، دارای خصیصه های checked می باشند که کنترل checkbox را تیک دار نمایش می دهد.

پس از ویرایش course assignment ها، باید هنگامی که سایت به صفحه ی Index برگردانده می شود، تغییرات اعمال شده را تست و بررسی نمایید. برای فراهم آوردن این امکان می بایست یک ستون به جدول مورد نظر در داخل آن صفحه اضافه نمایید. در این صورت نیازی به استفاده از شی ViewBag نیست، زیرا اطلاعاتی را که می خواهید نمایش دهید از قبل در خاصیت پیمایشی Courses موجودیت Instructor، که در حال ارسال آن به صفحه به عنوان model هستید، موجود می باشد.

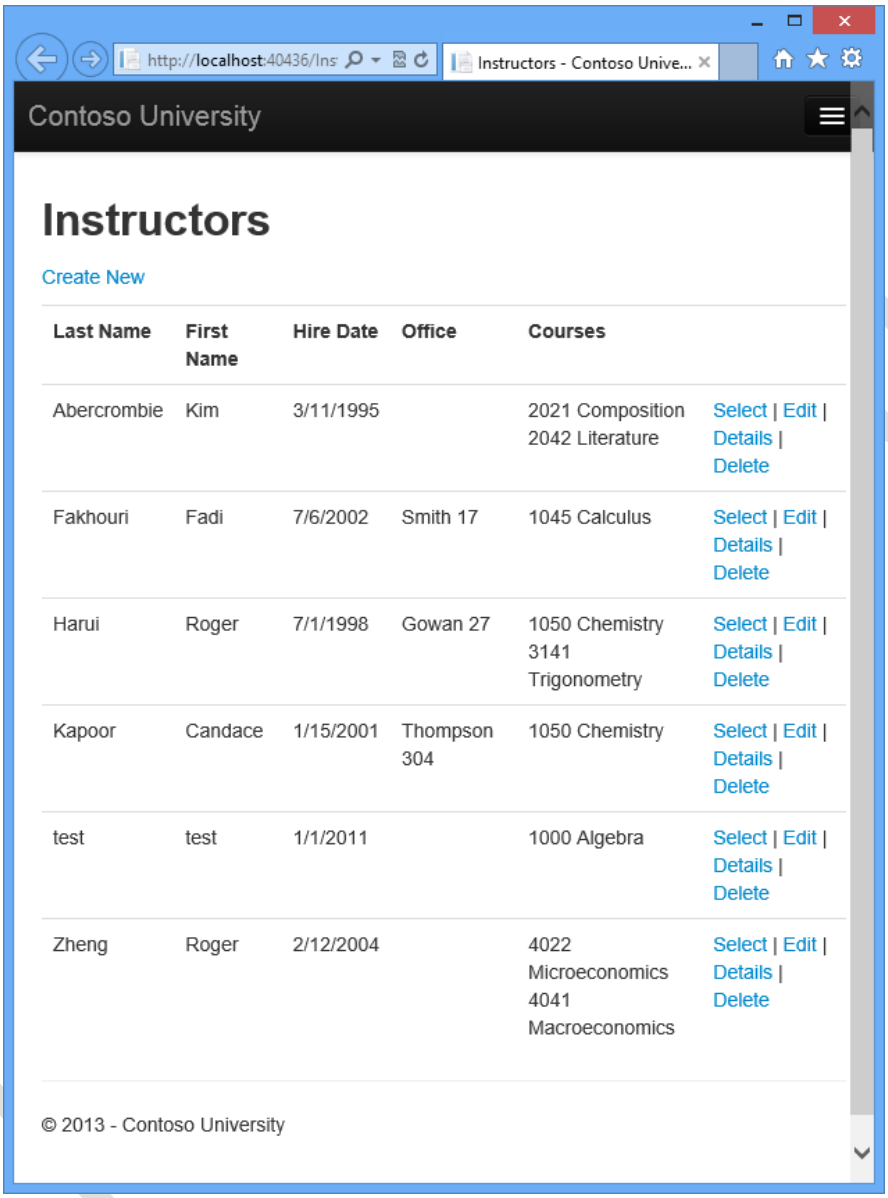
در فایل Views\Instructor\Index.cshtml، یک سرستون به نام Courses بلافاصله پس از سرستون ای که office نام دارد اضافه نمایید:

```
<tr>
 <th>Last Name</th>
 <th>First Name</th>
 <th>Hire Date</th>
 <th>Office</th>
 <th>Courses</th>
 <th></th>
</tr>
```

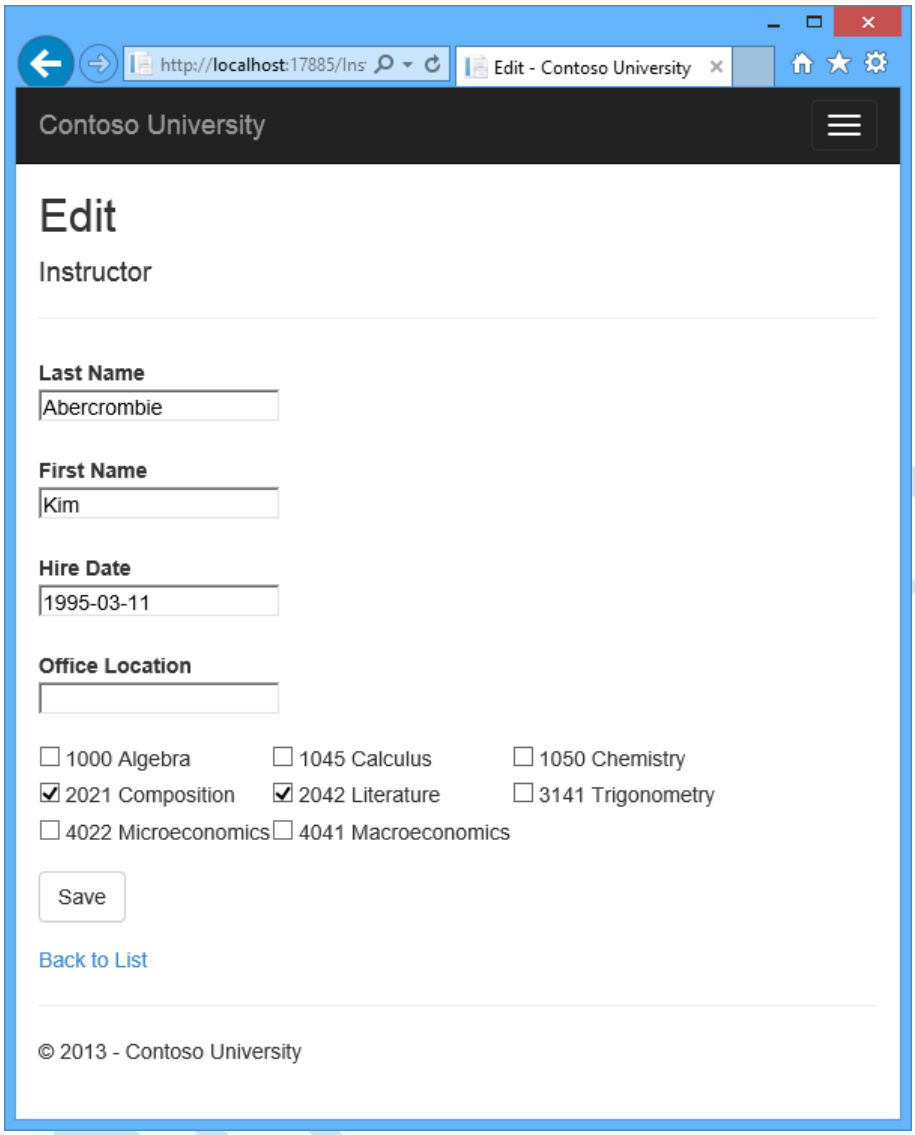
سپس یک detail cell دیگر، پس از office location detail cell اضافه کنید:

```
<td>
 @if (item.OfficeAssignment != null)
 {
 @item.OfficeAssignment.Location
 }
</td>
<td>
 @{
 foreach (var course in item.Courses)
 {
 @course.CourseID @: @course.Title <br />
 }
 }
</td>
<td>
 @Html.ActionLink("Select", "Index", new { id = item.ID }) |
 @Html.ActionLink("Edit", "Edit", new { id = item.ID }) |
 @Html.ActionLink("Details", "Details", new { id = item.ID }) |
 @Html.ActionLink("Delete", "Delete", new { id = item.ID })
</td>
```

صفحه ی Instructor Index را اجرا کرده تا course های اختصاص یافته به هر instructor، نمایش داده شود:



بر روی لینک Edit کلیک کرده تا صفحه ی Edit نمایش داده شود.



برخی از course assignment ها را تغییر داده و Save را کلیک کنید. تمامی تغییراتی را که ایجاد کرده اید در صفحه ی Index منعکس می شود.

نکته: روشی که در این آموزش برای ویرایش داده های instructor course استفاده شد، زمانی مناسب می باشد که تعداد course ها محدود باشد. برای مجموعه یا collection هایی که بسیار بزرگ تر هستند، به UI بسیار پیچیده تر و متد بروز رسانی متفاوت از آنچه در اینجا بکار برده شد، احتیاج است.

بروز آوری متد DeleteConfirmed

در فایل InstructorController.cs، متد DeleteConfirmed را حذف کرده و کد زیر را جایگزین آن کنید.

```
 [HttpPost, ActionName("Delete")]
[ValidateAntiForgeryToken]
public ActionResult DeleteConfirmed(int id)
{
  Instructor instructor = db.Instructors
    .Include(i => i.OfficeAssignment)
    .Where(i => i.ID == id)
    .Single();

  db.Instructors.Remove(instructor);

  var department = db.Departments
     .Where(d => d.InstructorID == id)
     .SingleOrDefault();
  if (department != null)
  {
     department.InstructorID = null;
  }

  db.SaveChanges();
  return RedirectToAction("Index");
}
```

کد بالا تغییرات زیر را ایجاد می کند:

. اگر Instructor مورد نظر به عنوان administrator یک department تخصیص داده شده باشد، در آن صورت instructor assignment را از آن department حذف می کند. بدون این کد، اگر سعی می کردید یک instructor را که به عنوان administrator تخصیص داده شده حذف کنید، در آن صورت یک خطای جامعیت ارجاعی (referential integrity) صادر می شد.

این کد مناسب سناریوهایی که در آن یک instructor به عنوان administrator به چندین department اختصاص می یابد، مناسب نبوده و قادر به مدیریت آن نیست.

افزودن office location و courses به صفحه ی Create

داخل فایل InstructorController.cs، متدهای HttpGet و HttpPost Create را حذف کرده، سپس کد زیر را جایگزین آن نمایید:

```
 public ActionResult Create()
{
  var instructor = new Instructor();
```

```
    instructor.Courses = new List<Course>();
    PopulateAssignedCourseData(instructor);
    return View();
}

[HttpPost]
[ValidateAntiForgeryToken]
public ActionResult Create([Bind(Include = "LastName,FirstMidName,HireDate,OfficeAssignment" )]Instructor
instructor, string[] selectedCourses)
{
    if (selectedCourses != null)
    {
        instructor.Courses = new List<Course>();
        foreach (var course in selectedCourses)
        {
            var courseToAdd = db.Courses.Find(int.Parse(course));
            instructor.Courses.Add(courseToAdd);
        }
    }
    if (ModelState.IsValid)
    {
        db.Instructors.Add(instructor);
        db.SaveChanges();
        return RedirectToAction("Index");
    }
    PopulateAssignedCourseData(instructor);
    return View(instructor);
}
```

این کد مشابه کدی است که در متدهای Edit مشاهده کردید، با این تفاوت که در ابتدا هیچ course ای انتخاب نشده است. متد HttpGet Create، متد PopulateAssignedCourseData را نه به این خاطر که course ای انتخاب شده، بلکه به منظور ارائه ی یک مجموعه ی خالی در view برای حلقه ی foreach در view، فرامی خواند (در غیر این صورت view code یک خطای null reference صادر می کرد).

متد HttpPost Create هر یک از checkbox های تیک دار course را به خاصیت پیمایشی Courses پیش از کد template اضافه می کند. کد template خطاهای اعتبارسنجی را چک کرده، سپس instructor جدید را به پایگاه داده اضافه می کند. حتی درصورت وجود خطاهای مربوط به model، course ها اضافه می شوند، تا زمانی که واقعا با خطاهای model مواجه شدیم (برای مثال، کاربر یک تاریخ غیرمجاز را وارد کرد) و صفحه مجدد با پیغام خطا نمایش داده شد، تمامی course هایی که انتخاب شده بودند، به صورت خودکار بازیابی شوند.

توجه داشته باشید که برای اینکه بتوان course هایی را به خاصیت پیمایشی Courses (navigation property به نام courses) افزود، بایستی خاصیت مربوطه را به عنوان یک مجموعه ی تهی (empty collection) مقداردهی اولیه کنید (یک collection جدید ایجاد کنید تا بتوانید course ها را به آن navigation property اضافه کنید):

```
instructor.Courses = new List<Course>();
```

یا بجای انجام این عملیات در کد controller، می توان با تنظیم property getter (خصیصه ی get) به گونه ای که در صورت عدم وجود collection آن را به صورت اتوماتیک ایجاد کند، این کار را در Instructor model انجام دهید:

```
        private ICollection<Course> _courses;
    public virtual ICollection<Course> Courses
    {
      get
      {
        return _courses ?? (_courses = new List<Course>());
      }
      set
      {
        _courses = value;
      }
    }
```

اگر خاصیت Courses را بدین شیوه اصلاح کنید، در آن صورت می توانید کدی که خاصیت را به صورت صریح مقداردهی اولیه می کند را از controller حذف نمایید.

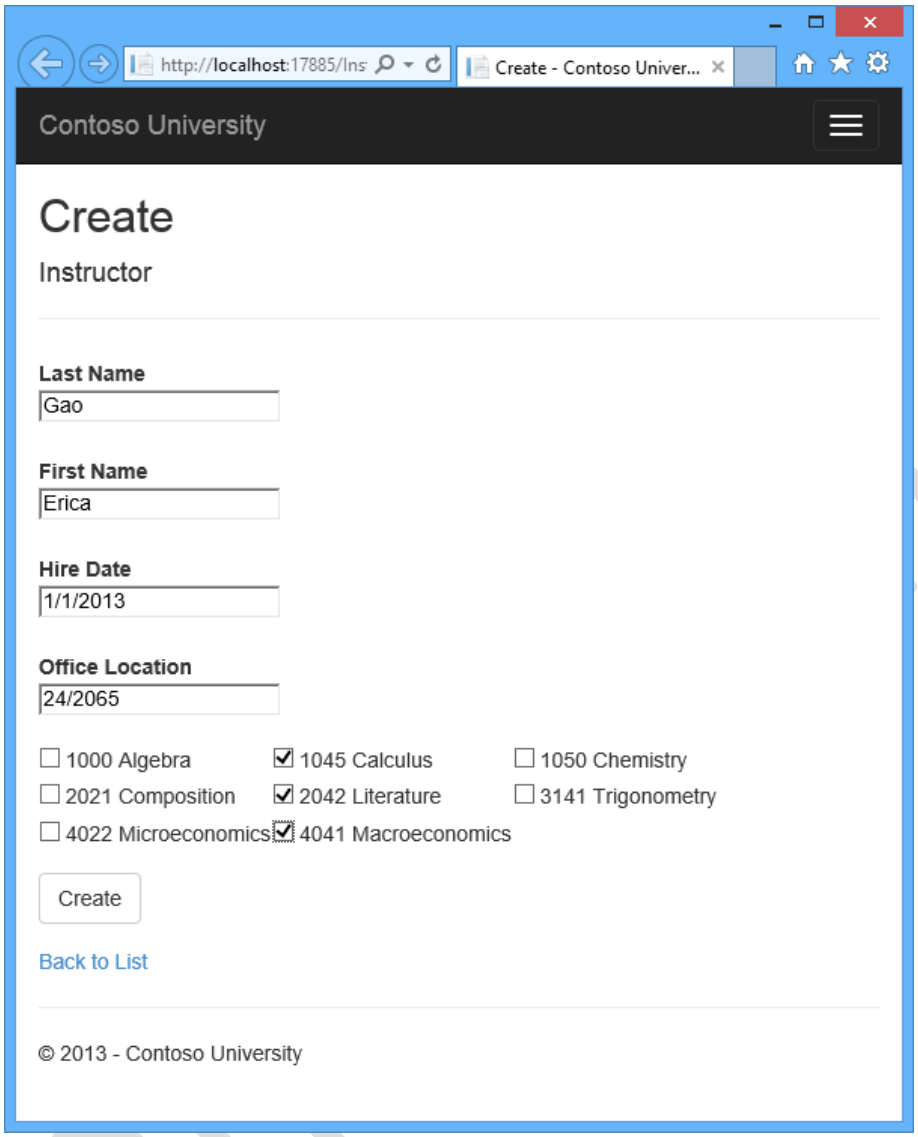
در فایل Views\Instructor\Create.cshtml، یک کادر متن (textbox) به نام office location اضافه کرده و کادرهای تیکی (checkbox) ویژه ی course ها پس از فیلد hire date و قبل از دکمه ی Submit اضافه کنید:

```
<div class="form-group">
  @Html.LabelFor(model => model.OfficeAssignment.Location, new { @class = "control-label col-md-2" })
  <div class="col-md-10">
    @Html.EditorFor(model => model.OfficeAssignment.Location)
    @Html.ValidationMessageFor(model => model.OfficeAssignment.Location)
  </div>
</div>
```

```
<div class="form-group">
  <div class="col-md-offset-2 col-md-10">
    <table>
      <tr>
        @{
          int cnt = 0;
          List<ContosoUniversity.ViewModels.AssignedCourseData> courses = ViewBag.Courses;

          foreach (var course in courses)
          {
            if (cnt++ % 3 == 0)
            {
              @:</tr><tr>
            }
            @:<td>
              <input type="checkbox"
                name="selectedCourses"
                value="@course.CourseID"
                @(Html.Raw(course.Assigned ? "checked=\"checked\"" : "")) />
                @course.CourseID @: @course.Title
            @:</td>
          }
          @:</tr>
        }
    </table>
  </div>
</div>
```

پس از جای گذاری (paste) کد، می بایست فاصله ی خطوط و توگذاری های را (که قبلا برای صفحه ی Edit تنظیم کردید) تنظیم کنید.

صفحه ی Create را اجرا کرده و یک instructor اضافه کنید.



به صورت پیش فرض، EF تراکنش ها را به صورت ضمنی پیاده سازی می کند.

آنچه در این مبحث انجام شد:

تاکنون تنها با کدهایی کار می کردیم که ورودی/خروجی (I/O) همزمان را مدیریت می کند. می توانید با پیاده سازی کد ناهمزمان (async code) کاری کنید که برنامه ی تحت وب به صورت کارآمدتر از منابع سرور وب استفاده کند. در فصل بعدی به شرح نحوه ی پیاده سازی و استفاده از کدهای ناهمزمان خواهیم پرداخت.