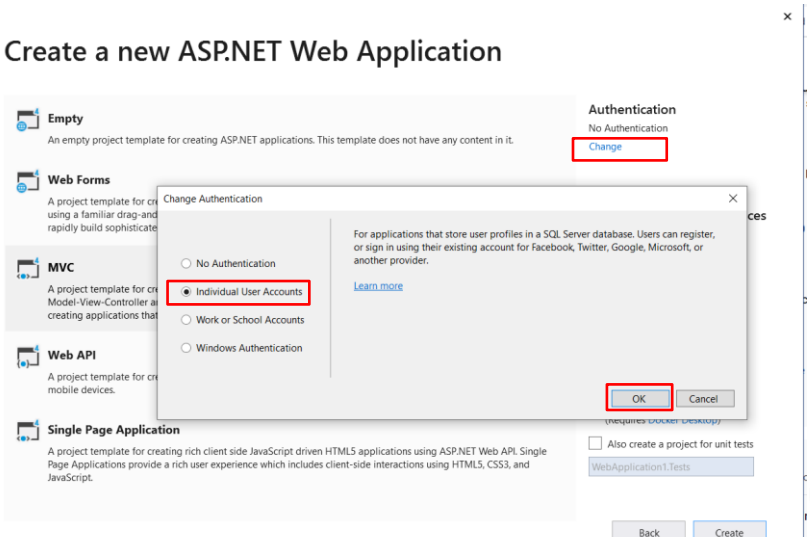
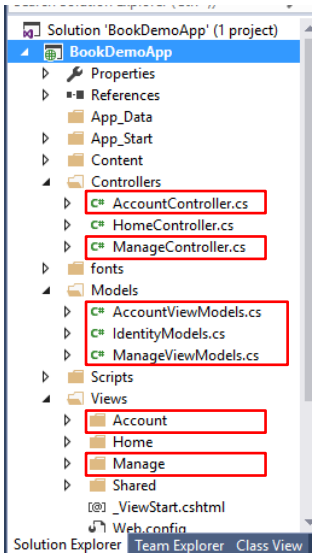
Authentication

Requiring Users to Log In

If we want our application to have authentication, we can set that up as we create the new project. Click Change under Authentication (see below), then choose Individual User Accounts.

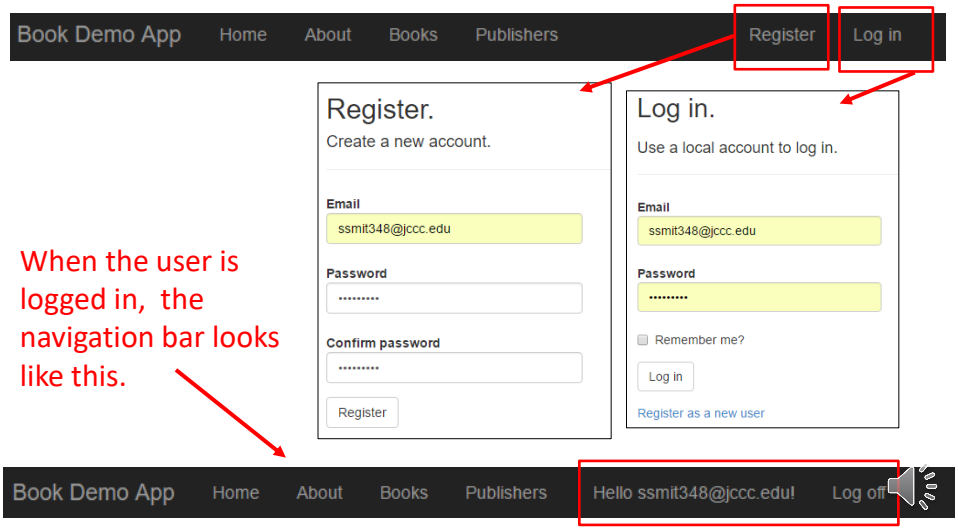


If you do this, you will notice some extra controllers, models, and views that have been created for you.



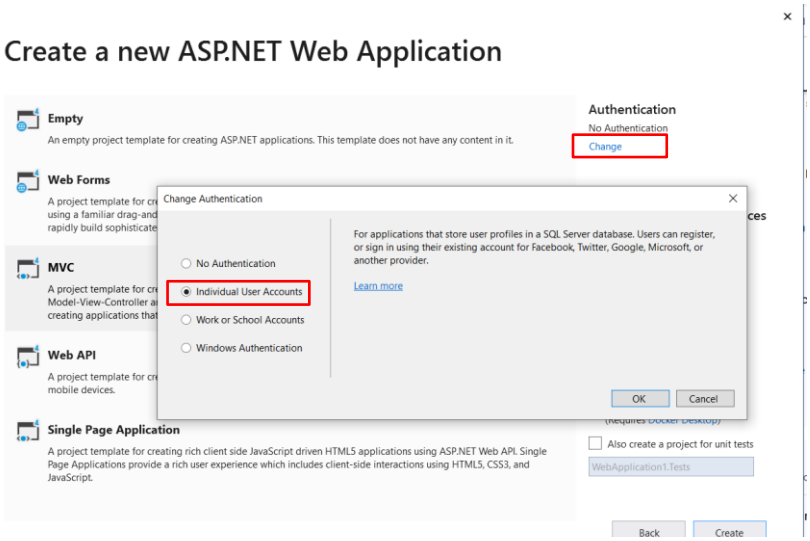
There are some new controllers, models, and views created for us since we chose **individual user accounts** for authentication.

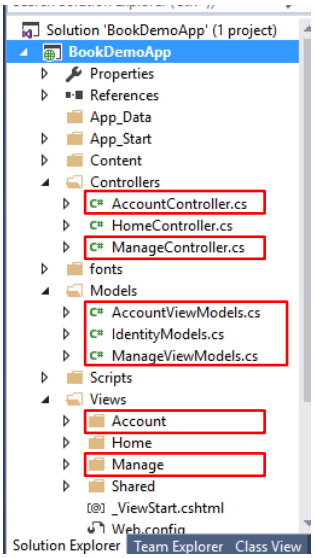
After you have gone through the steps to create a Code-First application, you'll find that two fully functional links will have been added for you on the right of the navigation bar. Everything is now in place to allow your users to log in or register for the first time if needed.



When the user is logged in, the navigation bar looks like this.

Let's create a new MVC application called BookDemoApp using Code First. Click the "Change Authentication" button and select "**Individual User Accounts**".





Notice, the extra controllers, models, and views that were created for us.

Add a Book class and a Publisher class in the Models folder.

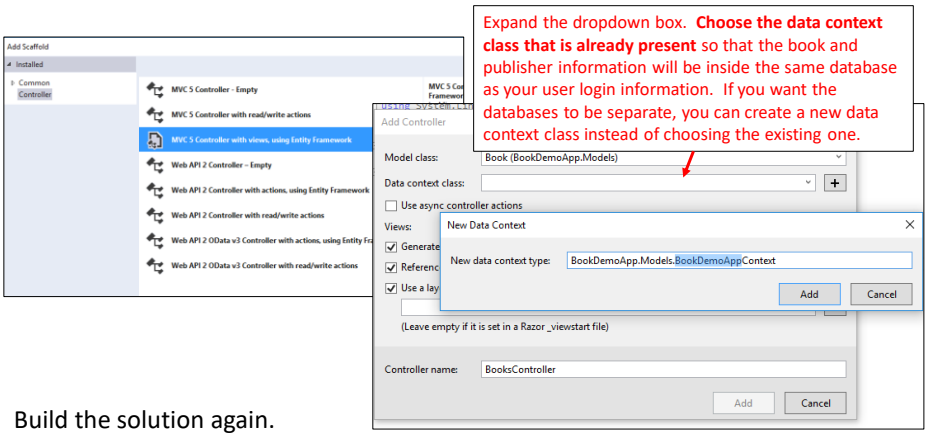
This code will be posted on Canvas if you'd like to copy and paste. Be sure though and verify that you understand the navigational properties that are there. Hopefully you could come up with them yourselves.

```
public class Book
{
    public int BookID { get; set; }
    public string Title { get; set; }
    public int PublisherID { get; set; }
    public decimal Price { get; set; }

    public virtual Publisher Publisher { get; set; }
}

public class Publisher
{
    public int PublisherID { get; set; }
    public string PublisherName { get; set; }

    public virtual ICollection<Book> Books { get; set; }
}
```

Build the solution, then add a **BooksController**.

Since authentication was set up when you created the project, there will **already be a data context class** in the dropdown box! (A database is needed to hold user information.)

If you want the Books table and the Publishers table to reside in the same database as the user logins, choose **that existing context class** for your new controller. This is usually the best option.

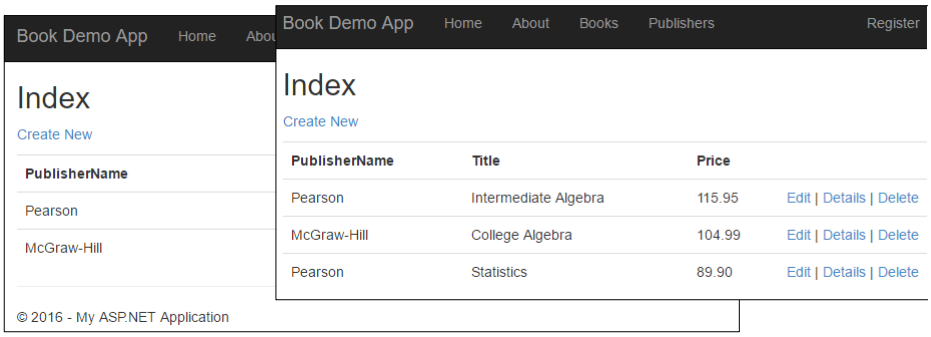


Build the solution again.
Add a **PublishersController**. For this new controller, use the data context used in the last step.

Change the home link text and add some links in _Layout.cshtml.

```
@Html.ActionLink("Book Demo App", "Index", "Home", new { area = "" },
                    new { @class = "navbar-brand" })
  <li>@Html.ActionLink("Books", "Index", "Books")</li>
  <li>@Html.ActionLink("Publishers", "Index", "Publishers")</li>
```

Run the application. Add two publishers and then three books.



Notice that need to change the **order of columns in the book list** and add a DisplayName attribute in the Publisher class. We won't take time to do that here.

Now it is easy to restrict a user from seeing a page unless they are logged in. Just put `[Authorize]` above the corresponding methods in the controller.

So let's say we don't want just anyone to be able to **create** a new **book** or a new **publisher**. We want to make sure that they are an authenticated user by seeing if they are logged in.

In that case, we would put [Authorize] above the **Create** methods in each controller.

```
// GET: Books/Create
[Authorize]
public ActionResult Create()
{
    ViewBag.PublisherID = new SelectList(db.Publishers, "PublisherID", "PublisherName");
    return View();
}
```

← **Add this!**

We would do the same for the Delete and Edit methods in each controller if we wanted those actions to be done by authenticated users only.

Note: There are **TWO** versions of the Create, Edit, and Delete methods in each controller: One for GET and another for POST. We'll talk about the difference between the two later in the semester.

For now, just know that you need to add [Authorize] to ***both*** *of these methods.*

Add [Authorize] above each version of a method.

```
// GET: Books/Create
[Authorize]
0 references
public ActionResult Create()
{
    ViewBag.PublisherID = new SelectList(db.Publishers, "PublisherID", "PublisherN
    return View();
}

// POST: Books/Create
// To protect from overposting attacks, please enable the specific properties you
// more details see https://go.microsoft.com/fwlink/?LinkId=317598.
[HttpPost]
[ValidateAntiForgeryToken]
[Authorize]
0 references
public ActionResult Create([Bind(Include = "BookID,Title,PublisherID,Price")] Book
{
    if (ModelState.IsValid)
    {
        db.Books.Add(book);
        db.SaveChanges();
        return RedirectToAction("Index");
    }
```

Application name

If a user is not logged in and they try to create a new book, they will be directed to the Log In page and once they log in, the requested page will be shown.

Index

Create New

PublisherName	Title	Price	
Pearson	Intermediate Algebra	115.00	Edit | Details | Delete
McGraw-Hill	Statistics	87.00	Edit | Details | Delete

© 2016 - My ASP.NET Application

What if you want to restrict access to all (or most) of your pages?

```
[Authorize]
public class BooksController : Controller
{
    private ApplicationDbContext db = new ApplicationDbContext();

    // GET: Books
    [AllowAnonymous]
    public ActionResult Index()

        var books = db.Books.Include(b => b.Publisher);
        return View(books.ToList());
    }
```

You can put **[Authorize]** above the ***entire*** **controller class.**

Put this in front of any method where authorization is not needed.

Then if there happens to any methods within that controller that you don't want authorized, put **[AllowAnonymous]** above the methods corresponding to those pages that you want **all users** to see.

What if you want only ***certain*** logged-in users to access a page?

Use this `[Authorize(Users="Alice,Bob")]`

Or we can create roles and specify them in the [Authorize] attribute.

```
// GET: Books/Create
[Authorize(Roles = "SalesManager")]
public ActionResult Create()
{
    ViewBag.PublisherID = new SelectList(db.Publishers, "PublisherID", "PublisherName")
    return View();
}
```

AspNetUsers

Id	Email	EmailConfirmed	PasswordHash	SecurityStamp	PhoneNumber	PhoneNumber...	TwoFactorEna...	L
40b6-a7dd-1721fac0f381	ssmit348@jccc.edu	False	ALpzCQK3DXo...	2f6db21d-f51b-...	NULL	False	False	N
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	N

AspNetRoles

Id	Name
1	SalesManager
NULL	NULL

AspNetUserRoles

UserId	RoleId
1ccee941-ede8-40b6-a7dd-1721fac0f381	1
NULL	NULL

There may actually be some views already created to allow an administrator to do this themselves through the application.

http://www.dotnetfunda.com/articles/show/2898/working-with-roles-in-aspnet-identity-for-mvc

https://code.msdn.microsoft.com/ASPNET-MVC-5-Security-And-44cbdb97

FYI:

ASP.NET MVC offers you an easy way to allow users to log in using other accounts like Google or Facebook.

This takes a two step approach.

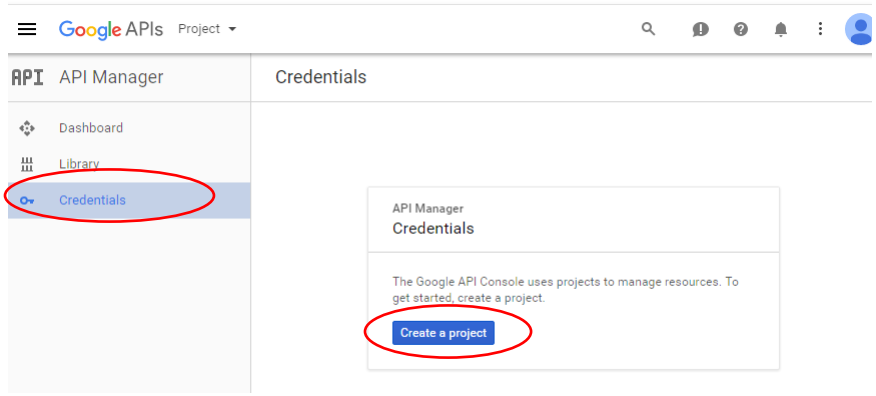
1. Register your application with the 3rd party through their website. You will receive a special ID and a password (usually called a Secret).

2. Find the commented code in Startup.Auth.cs (in the App_Start folder) specific to the 3rd party, uncomment it, and type in your ID and Secret.

To allow users to log in with their Google account:

Go to the Google Developers Console. You can find it by searching Google or click the link below.

Google Developers Console



New Project

Project name

ExternalLoginExample

Your project ID will be externalloginexample Edit

Show advanced options...

CANCEL CREATE

API key
Identifies your project using a simple API key to check quota and access.
For APIs like Google Translate.

OAuth client ID
Requests user consent so your app can access the user's data.
For APIs like Google Calendar.

Service account key
Enables server-to-server, app-level authentication using robot accounts.
For use with Google Cloud APIs.

Help me choose
Asks a few questions to help you decide which type of credential to use

API Manager

Dashboard
Library
Credentials

Credentials

Create client ID

To create an OAuth client ID, you must first set a product name on the consent screen

Configure consent screen

Application type
Web application
Android Learn more
Chrome App Learn more
iOS Learn more
PlayStation 4
Other

Google APIs ExternalLoginExample

API Manager

- Dashboard
- Library
- Credentials

Credentials

Create client ID

Application type

- Web application
- Android Learn more
- Chrome App Learn more
- iOS Learn more
- PlayStation 4
- Other

Name

ExternalLoginExample

Restrictions

Enter JavaScript origins, redirect URIs, or both

Authorized JavaScript origins

For use with requests from a browser. This is the origin URI of the client application. It can't contain a wildcard (http://*.example.com) or a path (http://example.com/subdir). If you're using a nonstandard port, you must include it in the origin URI.

https://localhost:44390 ←Instead of 44390, put the numbers that you see when you run your application in your browser.

http://www.example.com

Authorized redirect URIs

For use with requests from a web server. This is the path in your application that users are redirected to after they have authenticated with Google. The path will be appended with the authorization code for access. Must have a protocol. Cannot contain URL fragments or relative paths. Cannot be a public IP address.

https://localhost:44390/signin-google ←Instead of 44390, put the numbers that you see when you run your application in your browser.

Google APIs ExternalLoginExample

API Manager

- Dashboard
- Library
- Credentials

Credentials

Email address

ssmit348@jccc.edu

Product name shown to users

ExternalLoginExample

Homepage URL (Optional)

https:// or http://

Product logo URL (Optional)

http://www.example.com/logo.png

This is how your logo will look to end users
Max size: 120x120 px

Privacy policy URL
Optional until you deploy your app

https:// or http://

Terms of service URL (Optional)

https:// or http://

Save Cancel

The consent screen will be shown to users whenever you request access to their private data using your client ID. It will be shown for all applications registered in this project.

You must provide an email address and product name for OAuth to work.

OAuth client

Here is your client ID

173968995317-hrfr58hlk65vv9hdag9nh3p10m64gnec.apps.googleusercontent.co

Here is your client secret

olgcr7pYktIukw7tOTaMB0SJ

OK

API Manager

Dashboard

Library

Credentials

Library

Google Apps Marketplace SDK
Admin SDK
Contacts API
CalDAV API

Less

Mobile APIs
Google Cloud Messaging
Google Play Game Services
Google Play Developer API
Google Places API for Android

Social APIs
Google+ API
Blogger API
Google+ Pages API
Google+ Domains API

← Choose Google+ API and follow instructions to enable it.

YouTube APIs
YouTube Data API
YouTube Analytics API
YouTube Reporting API

Advertising APIs

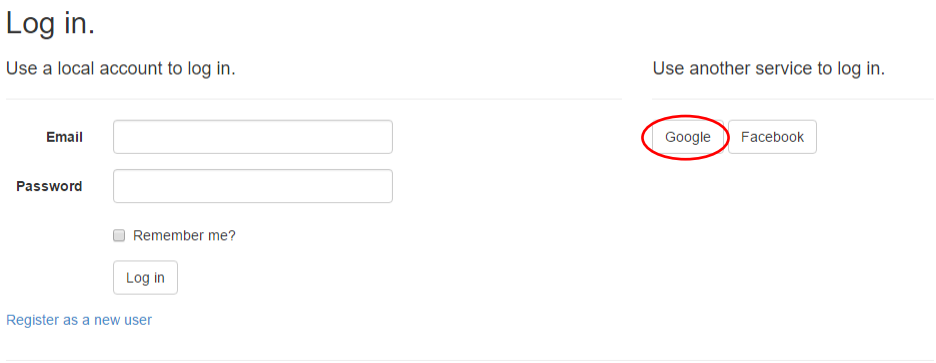
Now you need to put the ClientId and the ClientSecret into some code in your project.

Look in *App_Start\Startup.Auth.cs* and find this:

```
//app.UseGoogleAuthentication(new GoogleOAuth2AuthenticationOptions()
//{
//    ClientId = "",
//    ClientSecret = ""
//});
```

Take out the comment slashes and put in your App ID and Your App Secret.

```
app.UseGoogleAuthentication(new GoogleOAuth2AuthenticationOptions()
{
    ClientId = "173968995317-hrfr58hlk65vv9hdag9nh3p10m64gnec.apps.googleusercontei
    ClientSecret = "olgcr7pYktIukw7tOTaMB0SJ"
});
```



Authentication Lab

Create an MVC Code-First app with at least two entities that are related. Please **do not use** any entities that we have used in class examples or in assignments. If you need some ideas, here are some possibilities:

- Songs/Artists
- Games/Publishers
- Employees/Departments
- Houses/Realtors

Use Code First and allow for **individual user accounts**. There should be a Register link and a Login link on the navigation bar when a user first runs the application.

A user that has not logged in should not be able to change any of the information in the database. In other words, they should not have access to any pages that create, edit, or delete records. They should, however, be able to see the Index and Detail pages.

Enter at least 3 values for each entity.

Fix the code so that the columns are in the correct order for the pages that list the entities. Also fix any "Detail" page so that the information is in the correct order.

Create a video showing me that an unauthorized user cannot add an entity but that once the user logs in, they are able to do it. In the video, go through the process of adding one entity for each type.