**SEIS752**

Advanced Web Design

Spring 2015

**Homework 8**

MVC with Visual Studio

John Stark

May 21, 2015

**Outline**

1. Introduction
2. **My solution**
3. What is needed
   1. MySQL
   2. Visual Studio
4. **Evaluation of Visual Studio and MVC5 APS.NET C#**
5. Appendices
6. My notes as I developed this solution
7. **Introduction**

This assignment is to implement the FF solution in C# using Visual Studio and MVC5. The perceived purpose of the assignment is to gain experience with this development environment and to consider the advantages and disadvantages it offers the developer. Instructions were provided that guided the student through the set-up of a ‘boiler plate’ MVC project. The student must then add the FF functions, pages, etc. to complete the assignment.

I have added some notes in appendix A for the purpose of conveying some sense of what is involved. A quick skim of this appendix should be sufficient to convince the reader that there is a fair amount of complexity involved is the set up. These rough notes were collected along the way, and are not intended to be read word-for-word.

1. **My Solution**

I followed the instructions provided and created a basic MVC project. In general terms, I then made a copy of the *details* view and saved it as *FindNeighbors*. I implemented the FF functions on the *FindNeighbors* view. This way my changes would not break the working pages of the basic project.

Now for some details. I started by copying *Views/Users/Details.cshtml* and saving it as *FindNeighbors.cshtml* in the same directory. I added a link to *FindNeighbors.cshtml* on the *Index.cshtml* view. The homework instructions said to add a list of FF users that are located within one mile the subject of the details view. So now my new *FindNeighbors* view had the details portion.

Next I needed to a list to the *FindNeighbors* view. As a starting point I copied from the *Index* view which had a working list. But when I added the relevant code, it broke the *FindNeighbors* view. I spent two evenings (about 5 hours) trying to get both the list and details to co-exist on the view *FindNeighbors* without success. The problem seemed to have something to do with adding:

@model IEnumerable<SEIS752\_MVC\_WebApp\_stark.Models.FFUser>

In the interest of making some progress on adding the function to find FF users within one mile, I removed the *Details* code from the *FindNeighbors* view. I modified the *Index* view so that it would pass latitude and l;ongitude of the selected user to the *FindNeighbors* view. Using the debug execution mode in Visual Studio I was able to see the parameters passed with the URL.

@Html.ActionLink("FindNeighbors", "FindNeighbors",

new { \_lat=item.lat, \_lon=item.lon })

I spent 3 more evenings (about 9 hours) figuring out that I need to modify *Controller/UserControllers.cs* to pull the data I wanted from the data base and pass it to the *FindNeighbors* view. I spent another couple evenings (5-6 hours) figuring out the correct code to achieve the desired functionality.

// GET: /User/FindNeighbors

// reference for the query is:

// h...p://chsakell.com/2013/08/24/retrieving-data-with-dbcontext/

// start simple – just try to get users with lat less than what was

// passed in. Once that works, build a more complex query

public ActionResult FindNeighbors(double? \_lat, double? \_lon)

{

// return View(db.FFUsers.ToList());

var query = (from u in db.FFUsers

where (u.lat < \_lat + 1) && (u.lat > \_lat - 1)

&& (u.lon < \_lon +1) && (u.lon > \_lon -1)

orderby u.lat

select u).ToList();

var q = query.ToList();

List<FFUser> areClose = new List<FFUser>();

areClose.Add(q[1]);

double LAT; double LON; double miles;

for (int i = 0; i < q.Count; i++)

{

LAT = q[i].lat;

LON = q[i].lon;

miles = distanceTo( \_lat, \_lon, LAT, LON);

if( miles < 1){

areClose.Add(q[i]);

}

}

return View(areClose);

}

private double distanceTo(double? \_lat, double? \_lon,

double? LAT, double? LON)

{

double rlat1 = (double) (Math.PI \* \_lat / 180);

double rlat2 = (double) (Math.PI \* LAT / 180);

double rlon1 = (double) (Math.PI \* \_lon / 180);

double rlon2 = (double) (Math.PI \* LON / 180);

double theta = (double) (\_lon - LON);

double rtheta = (double) (Math.PI \* theta / 180);

double dist = Math.Sin(rlat1) \* Math.Sin(rlat2) +

Math.Cos(rlat1) \* Math.Cos(rlat2) \* Math.Cos(rtheta);

dist = Math.Acos(dist);

dist = dist \* 180 / Math.PI;

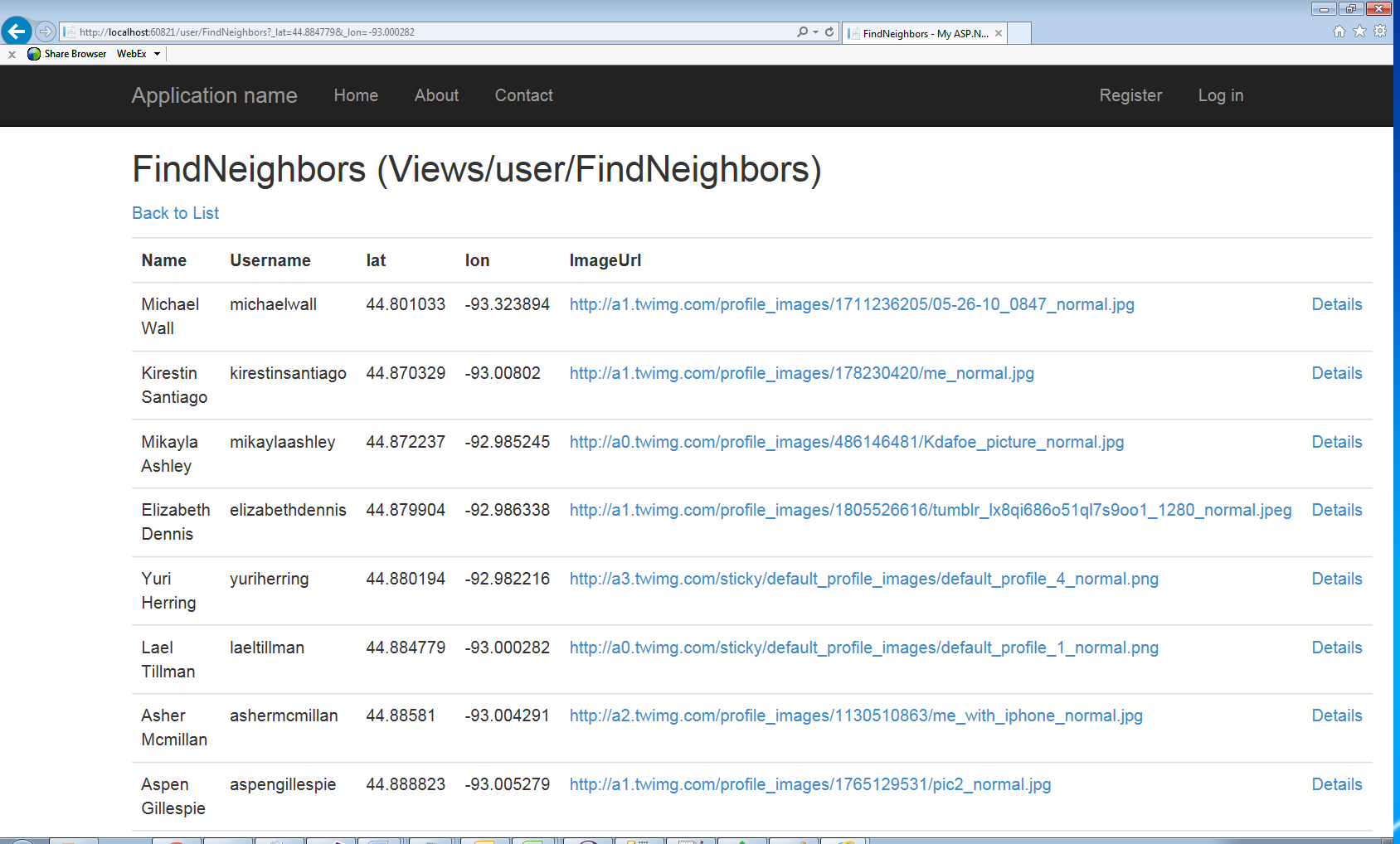
dist = dist \* 60 \* 1.1515;

dist = dist \* 0.8684;

return dist;

}

So, my *FindNeighbors* view presents a list of users within one mile of the users selected from the list on the *Index* view, as shown below. I counted the numbers of users in the *FindNeighbors* list to verify that a subset of all users was returned. I did not have time for more extensive testing/verification.



1. **What is needed**

For this assignment’s solution two applications need to be installed.

1. **MySQL**

I downloaded and installed the ‘community’ (free) version of MySQL and installed it. In fact I installed it 4 or 5 times trying to get it to work. After several hours, I eventually discovered that the installer from MySQL was installing a **.ini** file in the wrong location, and with a wrong name. Once that issue was resolved I was able to run the MySQL workbench and create a database and tables from there. This is an application I have used frequently in the past without issues, but the version downloaded for this assignment is a couple years newer.

1. **Visual Studio**

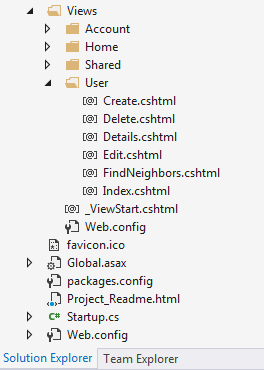
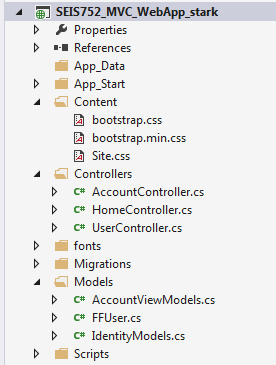
I already had Visual Studio installed on my work machine so I did not need to do this step. Note that a license is required to use Visual Studio. The one difficulty I had with Visual Studio was that I could not get it to connect to MySQL. I was able to work with the built in database to complete this assignment, but to deploy a solution a standalone database would be required.

1. **Evaluation of Visual Studio and MVC5 APS.NET C#**

The MVC solution with Visual Studio and MySQL offers some advantages but at a cost. Compared to a PHP/ MySQL / HTML solution, the complexity is an order of magnitude greater. That means that there are many more ways that things can go wrong. And it means that there is a longer and larger learning curve. All told, I spent about 4 evenings per week for 3 weeks to complete this assignment. That’s 3 weeks x 4 evenings x 3 hours per evening for a **total of 36 hours**. And a few parts of the solution don’t work yet! (note: I use Visual Studio mostly to connect with a TFS repository, so coding up a C# application was new to me.)

The partial view of the solution explorer pasted below gives some indication of the number of files involved. Compare this to a Java MVC solution that might have a s few as 3 files

My advice would be to avoid investing in this MVC Visual Studio solution solely for smaller or shorter duration projects. The overhead is burdensome. The learning curve could exceed the development time. And MicroSoft will require that you purchase license(s).



**In summary, cost, complexity and learning curve are the main disadvantages of the Visual Studio approach.**

If your staff is already proficient with MVC Visual Studio etc., then you might consider using it. If you have a large project, the advantages might provide some payback. Intertech in Eagan, MN offers a 40 hour course on Visual Studio MVC 5 ASP.NET so there are some people / organizations willing to invest in this technology, which implies that there are advantages.

Now to the advantages. This first that comes to mind is the same thing that frustrated me for so many hours – the framework is rigid and requires things be done a certain way. If I put a file in the wrong directory it wouldn’t compile (and note that a directory structure is provided). If I put a method in the wrong class it could not be accessed. In short the users are forced to adhere to the pattern provided. This enforces consistency; it prevents bad coding habits; and it makes the code easier to support. All of which are especially valuable for large teams of developers and code supported over long periods of time.

Next, there are some built in efficiency aids like type ahead, and color coded text in the editor (although other IDE also offer this). There is the scaffolding feature that will generate some Model / View / Controller code. Classes and methods get stubbed in. All of this relieves the developer of mundane repetitive typing tasks, and produces a lot of functionality is a short time. To make the most of these feature, some training and experience is required.

**In summary, consistency, efficiency, and best coding practices are the main advantages.**  And the larger the project and or team, the greater benefit will be.

**Appendix A.**

There is no need to read this appendix - a quick skim will convey the message that using Visual Studio MVC is complex.

***Installation Using MySQL Installer***

The general MySQL Installer download is available at <http://dev.mysql.com/downloads/windows/installer/>. The MySQL Installer application can install, upgrade, and manage most MySQL products, including MySQL Workbench.

When executing [MySQL Installer](http://dev.mysql.com/doc/refman/5.6/en/mysql-installer.html), you may choose MySQL Workbench as one of the products to install. It is selected by default, and essentially executes the standalone Installer Package described below.

**Installation Using the Installer Package**

The standalone download is available at <http://dev.mysql.com/downloads/tools/workbench/>.

MySQL Workbench can be installed using the Windows Installer (**.msi**) installation package. The MSI package bears the name mysql-workbench-**community**-**version**-win**arch**.msi, where **version** indicates the MySQL Workbench version number, and **arch** the build architecture (either win32 or winx64).

1. To install MySQL Workbench, right-click the MSI file and select the Install item from the pop-up menu, or double-click the file.
2. In the Setup Type window you may choose a Complete or Custom installation. To use all features of MySQL Workbench choose the Complete option.
3. Unless you choose otherwise, MySQL Workbench is installed in C:\**%PROGRAMFILES%**\MySQL\MySQL Workbench 6.3 **edition\_type**\, where **%PROGRAMFILES%** is the default directory for programs for your locale. The **%PROGRAMFILES%** directory is defined as C:\Program Files\ on most systems.

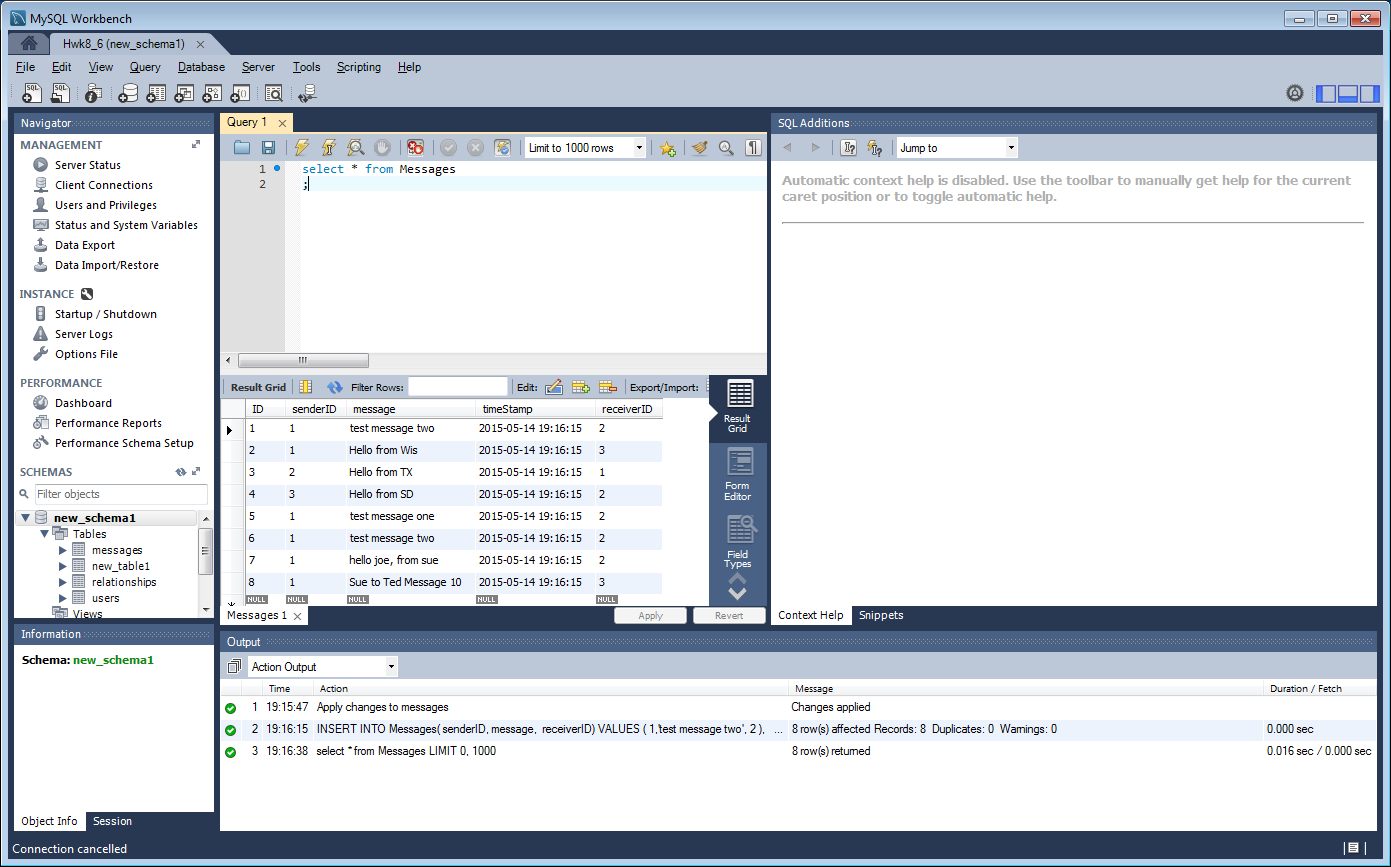
***Correcting the .ini file***

"C:/Program Files/MySQL/MySQL Server 5.6/bin\mysqld" --defaults-file="C:\ProgramData\MySQL\MySQL Server 5.6\my.ini"

C:\> cd "C:\Program Files\MySQL\MySQL Server 5.6\bin"

C:\> mysqld -nt --init-file=C:\\mysql-init.txt

***Created some demonstration tables in MySQL***



***Some password problems***

Create a text file containing the following statement on a single line. Replace the password with the password that you want to use.

SET PASSWORD FOR 'root'@'localhost' = PASSWORD('MyNewPass');

Save the file. This example names the file C:\mysql-init.txt.

Open a console window to get to the command prompt: From the Start menu, select Run, then enter cmd as the command to be run.

Start the MySQL server with the special --init-file option (notice that the backslash in the option value is doubled):

C:\> cd "C:\Program Files\MySQL\MySQL Server 5.0\bin"

C:\> mysqld-nt --init-file=C:\\mysql-init.txt

If you installed MySQL to a different location, adjust the cd command accordingly.

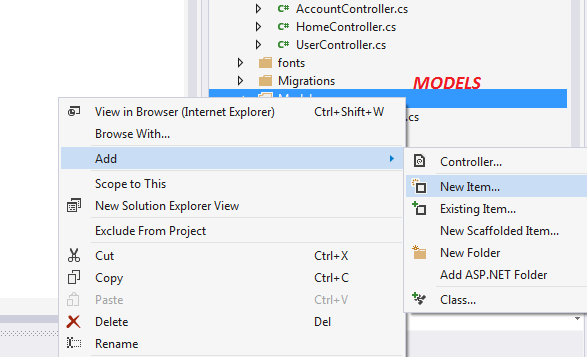
The server executes the contents of the file named by the --init-file option at startup, changing the 'root'@'localhost' account password.

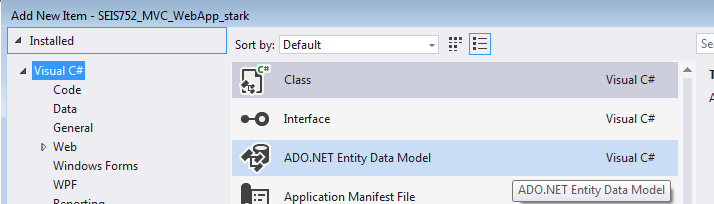
To have server output to appear in the console window rather than in a log file, add the --console option to the mysqld command.

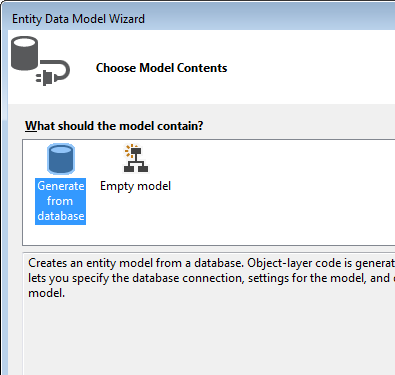
"C:\Program Files\Microsoft Policy Platform\policyHost.exe" /service

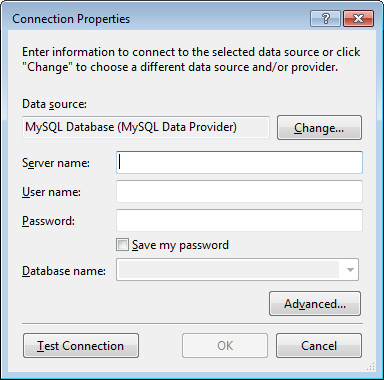
***Trying to connect Visual Studio to MySQL***

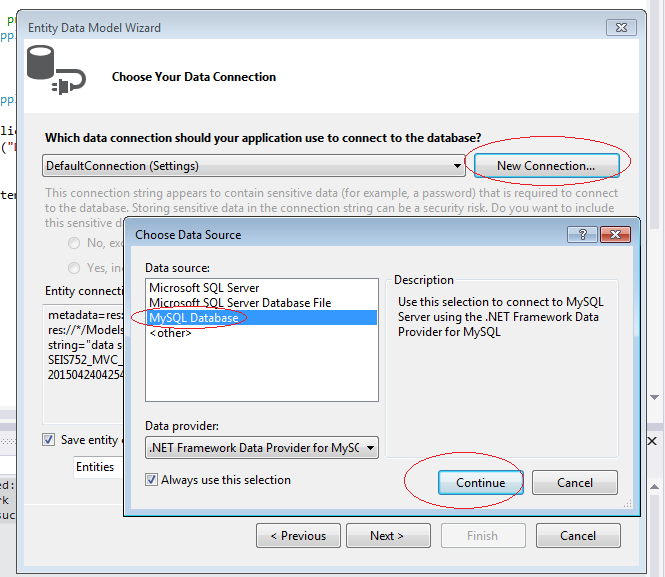
========== Rebuild All: 1 succeeded, 0 failed, 0 skipped ==========

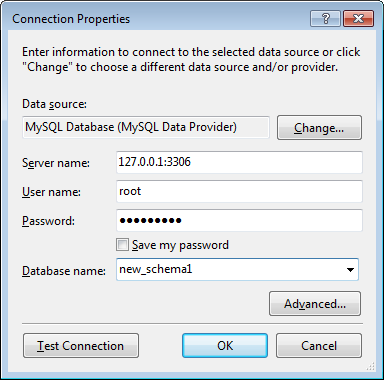


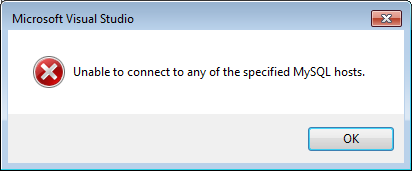


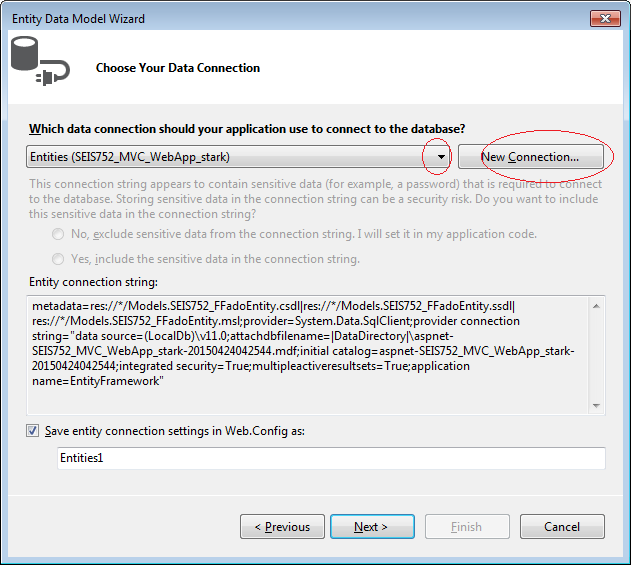












***Discovering how things in Visual Studio are connected***

