ASSIGNMENT #4 : MVC APPLICATION

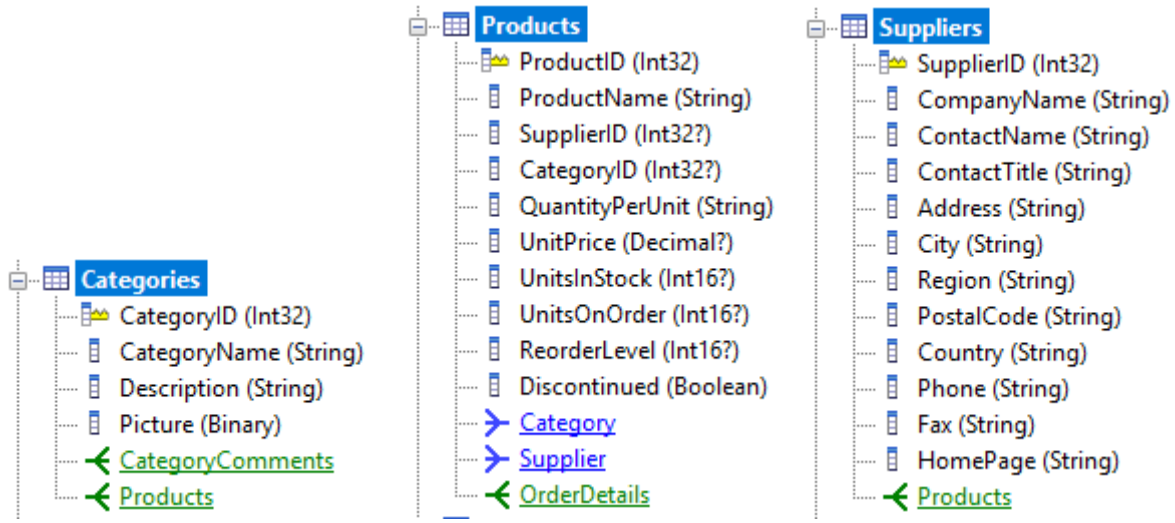
INTRODUCTION

This assignment requires an **MVC** service, called **MVC-SQL**, which retrieves data from three **SQL** tables, **Categories**, **Products** and **Suppliers**, from a slightly modified copy of the public **Northwind SQL** Server database sample, assumed to be located in the **C:\usertmp** local folder.

MVC-SQL returns human readable **HTML** pages containing a read-only server-side paginated grid view, which is directly accessible via a **browser**.

TABLE METADATA

A look the original SQL types, as Linqpad sees and maps them to C# (same as in A#3):



Note again the following **primary keys**: **CategoryID** for **Category**, **ProductID** for **Products**, and **SupplierID** for **Suppliers**. Note also the following **foreign keys**: **CategoryID** and **SupplierID** in **Products**. Together, these define the following "duck's paw" one-to-many and many-to-one relationships:

- **Products**: from **Categories** to **Products**, and **Category**: the other way round
- **Products**: from **Suppliers** to **Products**, and **Supplier**: the other way round

Our MVC application uses only part of these SQL columns:

- From **Categories**: **CategoryID**, and **CategoryName**.
- From **Products**: **ProductID**, **ProductName**, **SupplierID**, **CategoryID**, **UnitPrice**, **UnitsInStock**, and **UnitsOnOrder**. Note that the last 5 of these columns are **nullable** numbers!
- From **Suppliers**: **SupplierID**, **CompanyName**, **ContactName**, and **Country**.

The markers will initially use the same SQL database as given to you. The given **Products** table does NOT currently contain any **null** numerical values. However, for full marks, the markers will also test your service with an expanded version of the **Products** table, which will contain a few **null** numerical values.

CONTROL FLOW CYCLE - IN OUR MVC WEB APPLICATION

The **MVC-SQL** service offers a **REST** like callable controller method called via **Sql/WebGrid?**, which in the end returns an **HTML TABLE** with a page displaying the **LEFT JOIN** between table **Products** and tables **Categories** and **Suppliers**, plus a few auxiliary controls.

Note that an **INNER JOIN** will not give the same results, as it will skip the **Products** rows with no associated **Category** or **Supplier**.

1. The client (browser) sends an HTTP Request to **Sql/WebGrid?,** with the following four possible query string parameters (missing or wrong parameters get default values):
    a. **sortCol** – **string**, the primary sorting column, default **ProductID**
    b. **sortDir** – **string**, the sorting direction, ASC or DESC, default **ASC**
    c. **rowsPerPage** – **int**, the page size, default **10**
    d. **page** – **int**, the page number, default **1**
2. → The MVC library parses the URL and invokes your **SqlController.WebGrid**(…) method, which has four method parameters corresponding to the four possible query string parameters
    a. This controller method needs to fetch required database data using the VS generated **edmx** context and models and your own model
3. → Then, this controller method passes all gathered data to your view **WebGrid.cshtml**
4. → This view formats the data as an HTML table and sends it as a HTTP Response to the client (browser), plus a few auxiliary controls.

Examples

- URL:

  http://localhost:8181/Sql/WebGrid?page=10&rowsPerPage=5&sortCol=Country&sortDir=DESC

- Method call (after automatic parsing):

  SqlController.WebGrid (page: 10, rowsPerPage: 5, sortCol: “Country”, sortDir=”DESC”)

- URLs (equivalent by our default rules):

  http://localhost:8181/Sql/WebGrid

  http://localhost:8181/Sql/WebGrid?page=1&rowsPerPage=10&sortCol=ProductID&sortDir=ASC

- Method call (after automatic parsing):

  SqlController.WebGrid (page: 1, rowsPerPage: 10, sortCol: “ProductID”, sortDir=”ASC”)

- Resulting HTML tables: please see the following sample request/responses

HTTP RESPONSE – TABLE HEADERS AND SORTING

The response HTML table contains all the previously mentioned columns.

The header display names will contain extra spaces in combined words, e.g. the text "**Product ID**" for column **ProductID**.

The table can be **sorted** by clicking on any of the column headers, as usually alternating between **ascending** and **descending** sorting orders.

To avoid ambiguities, if the primary sorting is on any other column except **ProductID**, then the sorting will also use **ProductID** as a secondary criterion, in the same direction as the primary sort. For example, these are possible sorting criteria:

- **ProductID ASC**
- **ProductID DESC**
- **Country ASC, ProductID ASC**
- **Country DESC, ProductID DESC**

IMPLEMENTATION AND FUNCTIONAL REQUIREMENTS

High-level functional C# - no SQL! For this assignment, no SQL code should be present in any of your files. You must only use high-level functional C# code (as discussed in the lectures), which will be automatically translated to SQL.

Server-side sorting and pagination! For this assignment, all sorting and pagination must be done by the **SQL Server**! Your app must only get the rows and columns required to be displayed in the response HTML table!

The project must have the same folder structure as the demo/skeleton, the solution and project must be named **WebAplication1** (which may be the default), and the database must be located outside the project, in **C:\usertmp** – NOT in project's **App-Data** subfolder!

For this assignment you need to start from "scratch", i.e. by creating your own VS solution! Please do NOT take a solution partially created by others – this can detected and considered as plagiarism.

ADDITIONAL CHECKLIST – NOT COMPLETE, BUT PLEASE ENSURE THESE REQUIRED DETAILS, WHICH MAY NOT APPEAR IN THE DEMO/SKELETON

- Response HTML table with complete borders and all the required columns
- Header of primary sorted column has up/down arrows (e.g. Product ID ▲)
- Extra space in headers – if required in composed words (e.g. Product ID)
- Primary sorting possible on all columns, coming from all tables, Products, Categories, Suppliers
- Secondary sorting on ProductID, in the same direction as the primary sorting
- Reasonable column sizes s.t. headers always fit in one row, even with sorting arrows
- UnitPrice has currency formatting
- Numbers are right aligned
- Table footer has a pager with a complete set of usual options (first, previous, page numbers, next, last)
- The "Ordering by" label indicates the current primary sorting column and direction (e.g. ProductID ASC)
- The automatically generated SQL code is displayed in a bottom textarea box
- For the purpose of sorting, nulls – if any – are considered lower than any non-null value, so they appear first in ASC order
- Illegal column names and sorting orders are replaced by default values

Some of these requirements (but not all) are visible in the following sample request/responses (screenshots).

FURTHER DEVELOPMENT REQUIREMENTS

Please follow the following indications, which will ensure a fair marking on our COMPSCI lab machines.

- The services must be built in **C#**, with **Visual Studio 2015**, using the **Web Application** solution.
- If you wish, you can add UI specific CSS or JavaScript code in **WebGrid.cshtml.**
- Use an **MVC 5** template, with a VS generated **edmx** data context and model, based on the given database file, **Northwind.mdf** (assumed in **c:\usertmp**).
- You will need to add your own data **model** (for your own web grid table row); a custom controller called **SqlController**, with one method called **WebGrid**; and its corresponding view **Sql\WebGrid.cshtml**.
- For a compact coding, we recommend a version of **Dynamic LINQ**, e.g. as defined in **Dynamic.cs.**
- You also need to insert a new menu item, **SQL WebGrid**, in the default generated global layout, **_Layout.cshtml**.

There are several ways to build such a project. However, unless you are already familiar with VS, we suggest that you first start be recreating the demo/skeleton project from scratch, following the step-by-step guide described in the appendix.

Please talk to us if you wish to experiment with other possible implementations, e.g. Web API, F#, WebSharper… Our general advice is to first solve the problems exactly as indicated above, and only then try other approaches – small bonuses might be possible, but these need to be agreed before.

RUNNING AND TESTING

Similar to A#4, but using a browser as client. Although the project was initially build with VS, we prefer to further compile and test it from command-line – this is much faster!

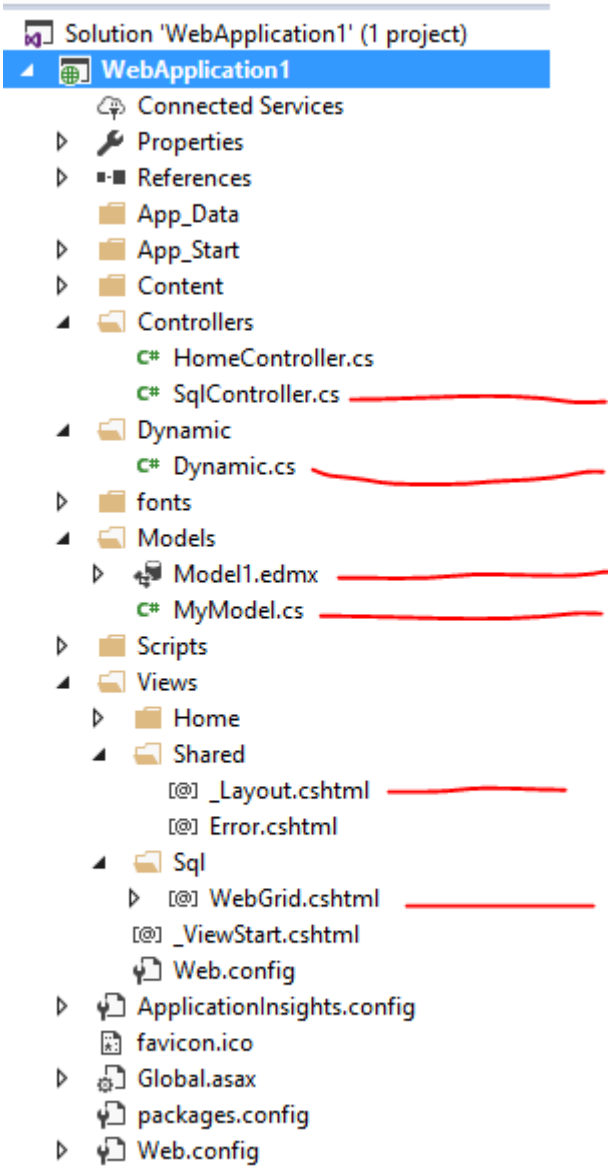
- To build the application: first run **_packages_restore.bat**, then **_msbuild-build.bat**
- To start the service (on 8181): run the appropriate version of **_Service_IISExpress_*.bat**
- To test the service: start the browser using the **_localhost-8181** url shortcut… then happy clicking
- To clean the package and bin folders: run **_packages_clean.bat** and **_msbuild-clean.bat**

Note: **VS** itself uses **msbuild**, but **msbuild** does NOT depend on **VS** … and is much faster!

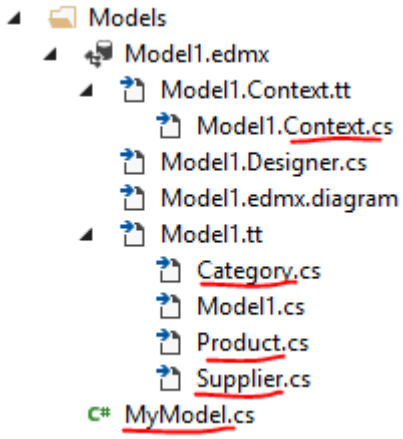
DEMO/SKELETON APPLICATION

We have a demo/skeleton application, called **mvc-sql-demo-minus**, which partially implements these specs for a couple of much smaller tables, **MyCustomers** and **MyOrders** – already known from A#3.

SUMMARY OF YOUR CONTRIBUTIONS (CREATED WITH HELP OR EVEN LOT OF HELP FROM VS WIZARDS)



MODEL DETAILS: **EDMX** CLASSES FOR CONTEXT (DATABASE), FOR REQUIRED TABLES, AND **OUR OWN MODEL**



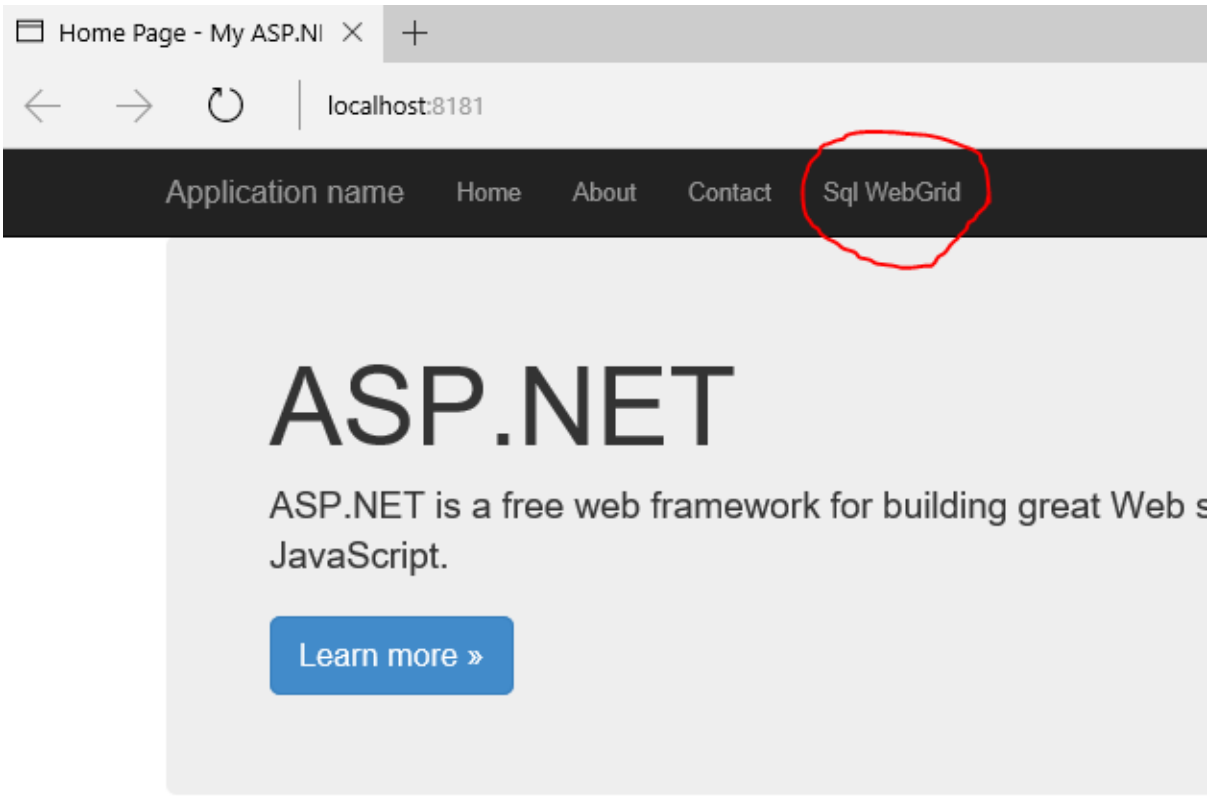
SAMPLE REQUEST/RESPONSE FRAGMENTS (BROWSER CLIENT)

Assuming that the service is available at port **8181** on the local machine

- **HTTP base request**: http://localhost:8181/

  Note your additional button, **Sql WebGrid**, on the default page (which was automatically generated by the VS MVC 5 template).

  Base response – our method SqlController.WebGrid is not yet involved

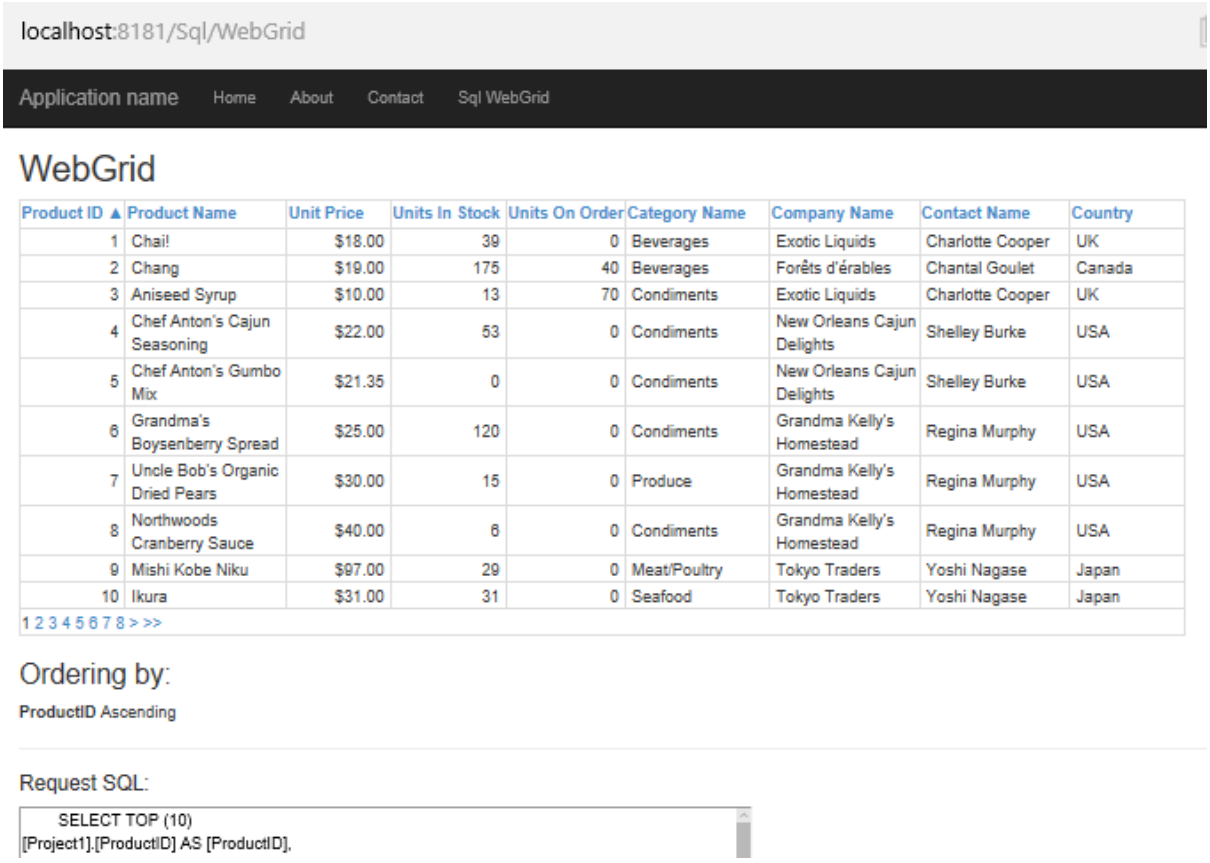


- **Sample HTTP Request**: http://localhost:8181/Sql/WebGrid

  This request takes all default parameter values – page: 1, …, so it is equivalent to

  http://localhost:8181/Sql/WebGrid?page=1&rowsPerPage=10&sortCol=ProductID&sortDir=ASC

  Response:



localhost:8181/Sql/WebGrid

Application name Home About Contact Sql WebGrid

WebGrid

Product ID ▲	Product Name	Unit Price	Units In Stock	Units On Order	Category Name	Company Name	Contact Name	Country
1	Chai!	$18.00	39	0	Beverages	Exotic Liquids	Charlotte Cooper	UK
2	Chang	$19.00	175	40	Beverages	Forêts d'érables	Chantal Goulet	Canada
3	Aniseed Syrup	$10.00	13	70	Condiments	Exotic Liquids	Charlotte Cooper	UK
4	Chef Anton's Cajun Seasoning	$22.00	53	0	Condiments	New Orleans Cajun Delights	Shelley Burke	USA
5	Chef Anton's Gumbo Mix	$21.35	0	0	Condiments	New Orleans Cajun Delights	Shelley Burke	USA
6	Grandma's Boysenberry Spread	$25.00	120	0	Condiments	Grandma Kelly's Homestead	Regina Murphy	USA
7	Uncle Bob's Organic Dried Pears	$30.00	15	0	Produce	Grandma Kelly's Homestead	Regina Murphy	USA
8	Northwoods Cranberry Sauce	$40.00	6	0	Condiments	Grandma Kelly's Homestead	Regina Murphy	USA
9	Mishi Kobe Niku	$97.00	29	0	Meat/Poultry	Tokyo Traders	Yoshi Nagase	Japan
10	Ikura	$31.00	31	0	Seafood	Tokyo Traders	Yoshi Nagase	Japan

1 2 3 4 5 6 7 8 > >>

Ordering by:

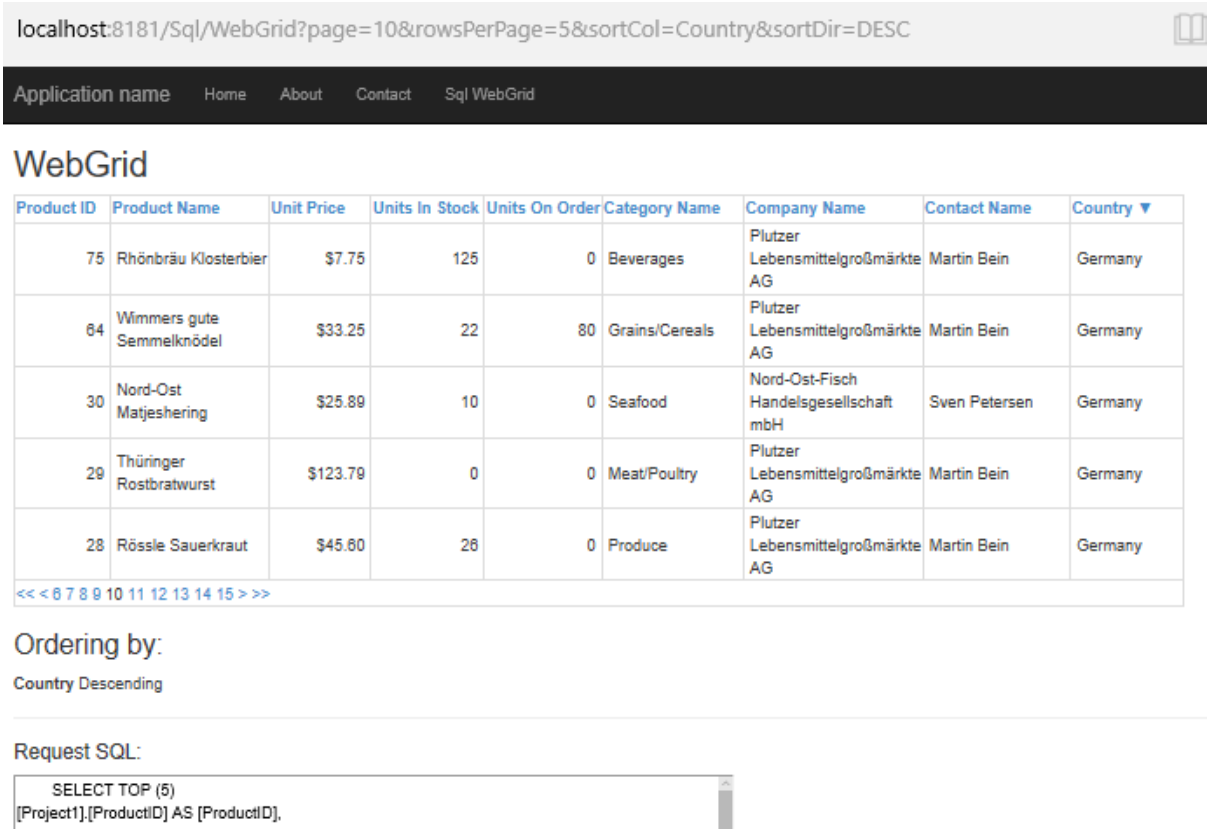
ProductID Ascending

Request SQL:

```
SELECT TOP (10)
[Project1].[ProductID] AS [ProductID],
```

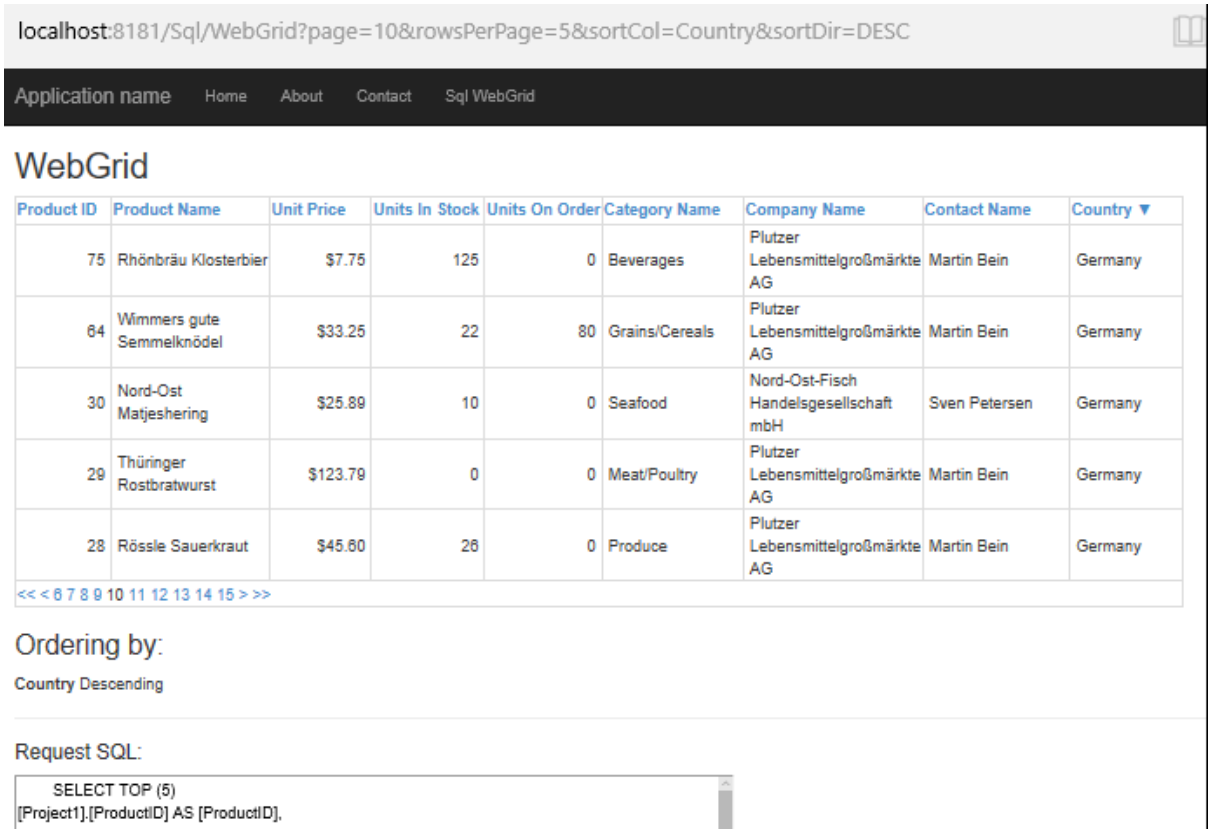
- **Sample HTTP Request**:
  http://localhost:8181/Sql/WebGrid?page=10&rowsPerPage=5&sortCol=Country&sortDir=DESC

  Response:



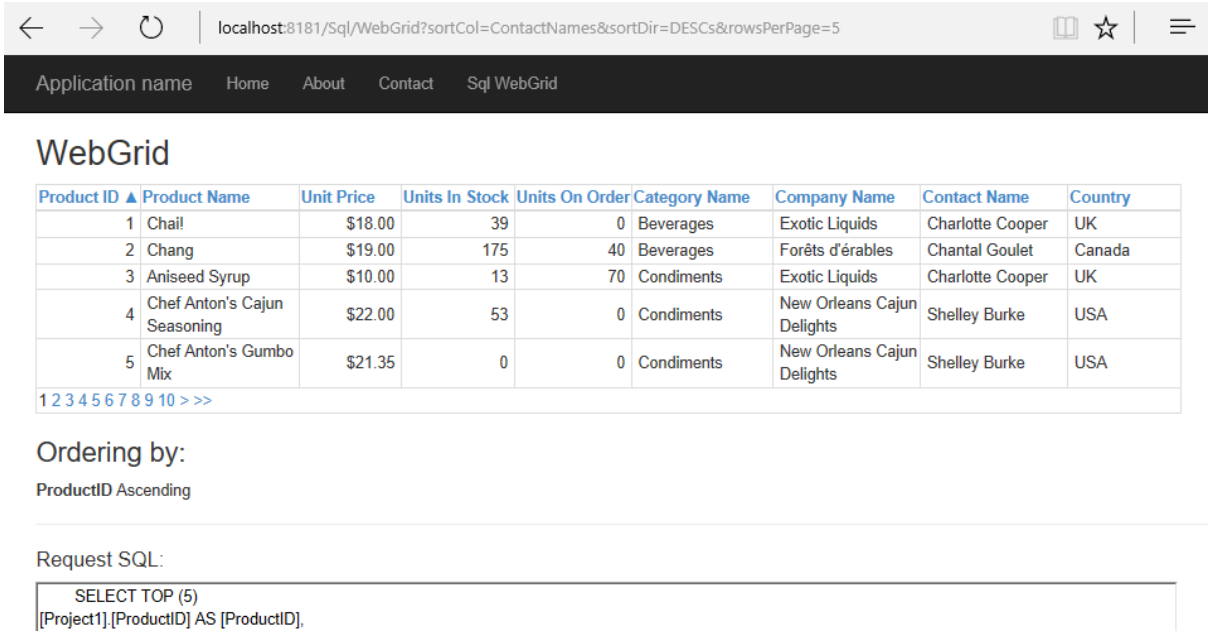
- **Sample HTTP Request – with custom rows per page**:
  http://localhost:8181/Sql/WebGrid?page=10&rowsPerPage=5&sortCol=Country&sortDir=DESC

  Response:



- **Sample HTTP Request – with incorrect column and sorting direction (replaced by defaults)**:
  http://localhost:8181/Sql/WebGrid?sortCol=ContactNames&sortDir=DESCs&rowsPerPage=5

  Response:



Additional Readings for MVC

- Getting Started with ASP.NET MVC 5
  http://www.asp.net/mvc/overview/getting-started/introduction/getting-started

  Many additional pointers available…
- GridView Class

  https://msdn.microsoft.com/en-us/library/system.web.ui.webcontrols.gridview(v=vs.110).aspx

Assignment Marks, Deliverables and Submission

This **C#** assignment is worth **5 total course marks**. Please note the strict COMPSCI policy on **plagiarism**.

Submit electronically, to the COMPSCI web dropbox, an **upi.7z** archive containing your **MVC-SQL-solution** folder, without its **bin**, **packages** and **App_Data** subfolders – or ensure that these are empty! These subfolders may be quite large and are not needed by the marker (which has identical copies). The submission size will be **limited to 2 MB** per student.

Please recheck your projects before submitting, starting from the archive prepared for submission (which is the only thing that the marker will receive from you). Expand this archive on a lab machine, in another location, then restore the **packages** folder, rebuild your solution and test it again!

You can submit several times, but only the **last submission** will be assessed. Please keep your electronic dropbox **receipt**!

Deadline

Wednesday 4 Oct 2017, 18:00! Please do not leave it for the last minute. Remember that you can resubmit and, by default, we only consider your **last submission**.

After this deadline, submissions will be still accepted for 4 more days, with gradually increasing penalties, of 0.5% for each hour late – as for A#3.

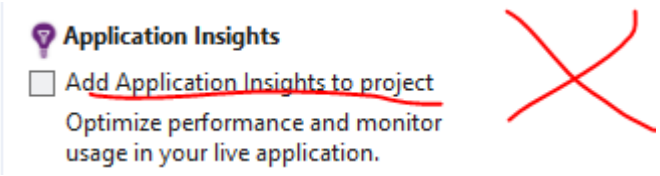
After this, no more submissions are accepted!

APPENDIX – RECREATING THE DEMO/SKELETON APPLICATION – STEP-BY-STEP GUIDELINE

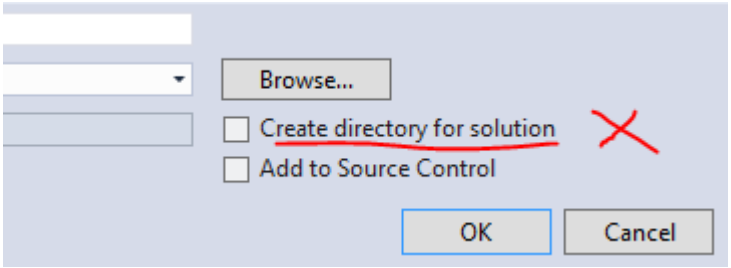
There are many valid ways, but we suggest that you first follow this apparently strict guideline.

EMPTY APPLICATION (with required templates and libraries)

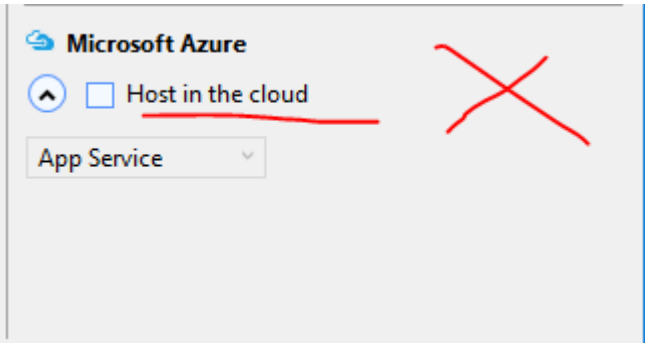
- ensure that our database is at its expected location **C:\usertmp\Northwind.mdf**
- start with an empty folder called **mvc-sql**
- start **VS 2015** and create a **new project**
- select **Visual C# … ASP.NET Web Application (.NET Framework)**
- name it **WebApplication1** (which is the usual default)
- browse to locate it in your new folder **mvc-sql**
- NO THANKS! uncheck VS's default offer to add Application Insights monitoring



- NO THANKS! uncheck VS's default offer to create (yet another nested) directory for the solution (otherwise the given batch files may NOT work)



- NO THANKS! uncheck/reject any offer to host the project in the Microsoft Azure cloud, e.g.



- select **MVC (MVC 5 Empty)**
- copy the following items from **demo/skeleton** into your **mvc-sql** folder: subfolder **.nuget**, subfolder **.packages** (even if empty), url **_localhost-8181**, and all **_*.bat** files
- wait until VS completes creating the project… and downloads/installs all required packages and templates (coffee break?)

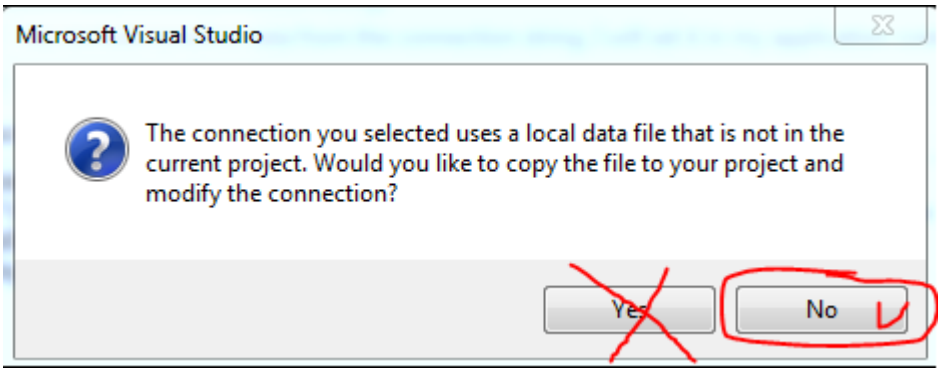
- now, prepare to use a lot of **right mouse clicks** – almost each time when there is no visible option, a right mouse click will put us back on the right path!

DYNAMIC LINQ

- using the **VS solution explorer**, hover over the **WebApplication1** project node and create a subfolder called **Dynamic** and drag in a copy of **Dynamic.cs** – this ensures that **Dynamic LINQ** will be automatically included in the compilation (otherwise we may need a few extra steps)

MODELS: NORTHWIND AND OUR OWN

- using the **VS solution explorer**, hover your mouse over subfolder **Model** and **add new item** … **Data** … **ADO.NET Entity Data Model (EDM)**
- using the **EDM wizard** configure this via **EF Designer from database** … **new connection** … **MS SQL Server Database File** … browse to select **C:\usertmp\Northwind.mdf** … **test connection** … **OK**
- NO THANKS to VS's attempt to make a local copy of it (into your solution's **WebApplication1.App_Data** subfolder, which must for us remain empty)



- still using the **EDM wizard** select our tables (**MyCustomers** and **MyOrders** for the **demo** replica)
- OK – dismiss any security warnings, if any
- wait again until VC creates the **Northwind** associated **edmx** bundle
- using the **VS solution explorer**, hover again your mouse over subfolder **Model** and **add new item** … **Class** … and name it **MyModel.cs**
- fill **MyModel** class as required, e.g. copy in the corresponding text from the **demo/skeleton**

CONTROLLER

- hover the mouse over **Controllers** and create a new **MVC 5 Controller (Empty)** named **SqlController**
- implement the **WebGrid** method by copy/paste from the **demo**
- you may notice a few red wiggled lines for unknown items
- add a **using** clause for the namespace containing **MyModel**, probably **WebApplication1.Models** – VS will help, if you ask via a right mouse click
- you may need to change the **name** of **NorthwindEntities1** to **NorthwindEntities** - check the exact spelling in the **edmx** context file, probably called **Model1.Context.cs**
- manually add a **using** clause for the **Dynamic LINQ** **OrderBy** (…) call
- your application should now compile, but won't run as expected, because we still need the view

VIEWS

- on **SqlController.WebGrid** code, hover the mouse over **return View(res)** and create a corresponding View, named **WebGrid** – which will be stored in subfolder **Views\Sql**
- to simplify, let it be **Empty (without model)** – we'll manually fix this by copy/pasting the corresponding **demo** view
- thus paste and copy the whole text of the demo **WebGrid.cshtml** – including the first line which declares the view model:

```
@model IEnumerable<WebApplication1.Models.MyModel>
```

- we only need one more item: a new **menu item** to access our controller and view
- open **Views\Shared_Layout.cshtml**, locate the **navigation bar** and insert this line:

```
<li>@Html.ActionLink("Sql WebGrid", "WebGrid", "Sql")</li>
```

These view use **Razor** syntax, a sharp (?) mixture of **C#**, **HTML** and **CSS** (and possibly some **JavaScript**, if you wish so…)

Hope that this guideline solves most of the problems faced by students who don't have experience with VS

We'll also demonstrate these steps in class, in the first week after the break (or even in one of the lectures, time permitting)