Seperation of Concerns

"[Separation of Concerns is the] idea that a software system must be decomposed into parts that overlap in functionality as little as possible."
https://effectivesoftwaredesign.com/2012/02/05/separation-of-concerns/

The **separation of concerns principle** says that software modules (Files, Classes, Methods) should be broken up such that each module has the smallest number of **concerns** possible. A concern has been defined a lot of different ways, by a lot of different people, but is often defined as **a reason for change.**

Thus… Every part of your code should have the fewest possible reasons to change. And a change to any part of your code should have a minimal impact to any other part.

"Since the first software systems were implemented, it was understood that it was important for them to be modular. It is necessary to follow a methodology when decomposing a system into modules and this is generally done by focusing on the software quality metrics of coupling and cohesion [...]:

Coupling: The degree of dependency between two modules. We always want low coupling.

Cohesion: The measure of how strongly-related is the set of functions performed by a module. We always want high cohesion.

All methodologies try to reduce coupling and increase cohesion."

https://effectivesoftwaredesign.com/2012/02/05/separation-of-concerns/

"Imagine you have a program which has been released…."

https://softwareengineering.stackexchange.com/a/32614/343223

Is separation of concerns being applied vertically or horizontally?

Resume Model	Skill Model	Project Model
Resume Controller	Skill Controller	Project Controller
Resume Views	Skill Views	Project Views

Is this a good example of separation of concerns?

View
Controller
Model
Entity Framework
ADO.NET
SQL Server

What are the benefits of splitting up work along vertical slices or horizontal slices?

Entity Framework

Entity Framework is an **Object-Relational-Mapper (ORM)** created by Microsoft which integrates well with Microsoft SQL Server, but can also be used with other **Database Management Systems (DMBS).**

Object-Relational-Mappers provide a strictly-typed, object-oriented interface to a database.

Benefits Include

- ❏ They hide many of the differences in SQL dialects between DBMSs and thus make it easier to transition to a different DMBS.

- ❏ They automatically enforce some constraints on the data, so that developers spend less time writing code to validate the data. And this also reduces the number of developer errors in failing to validate the data.

- ❏ They allow developers to continue working in their preferred high-level languages instead of building SQL queries as long strings, with little to no syntax highlighting.

- ❏ Since developers are not directly writing SQL it avoids the most common causes of SQL injection vulnerabilities.

Popular ORMs

- ❏ C# - Entity Framework
- ❏ Java - Hibernate
- ❏ Node.js - Mongoose

Model Attributes

[Table]	Used to define the name of a database table. (Instead of letting Entity Framework guess.)
[Key]	Used to define the primary key of a database table. (Instead of letting Entity Framework guess.)
[Column]	Used to define the name and/or order of columns in the database.

[Display]	Used to set the name that will be shown with LabelFor() / DisplayNameFor()
[DisplayFormat]	Used to set the format string that used to display/edit the column
[DataType]	Used to change what kind of editor EditorFor generates

[Required]	Column is required. (Still required if null isn't allowed)
[StringLength]	Defines the maximum length of a string.
[MinLength]	Defines the minimum length of a string.
[Regex]	Defines a pattern that a string must follow.

Navigation Properties

Navigation properties provide a simple way to retrieve related data from other tables.

Navigation properties should be marked virtual. Remember that virtual properties/methods can be overridden in a subclass. Entity framework dynamically creates subclasses of your model classes and this is necessary to support lazy loading.

One-to-many relationships can often be talked about in terms of parents and children. A parent has many children and a child has one parent. In this way the parent table should have a navigation property of type **IList<Child>** and the child should have a navigation property of type **Parent.**

Using Asynchronous LINQ Methods

Much of the work a web application does, relies on data from the database. Unfortunately accessing the database takes a good amount of time (10 milliseconds to 5 minutes). If we make the web server wait for data to be retrieved or saved, then it cannot serve other customers while it waits. This leads to a much slower user experience.

To avoid this, we need to enable the web server to handle other customers in the interim.

The **await** keyword tells ASP.NET to put our current job to sleep and to come back to us when the database is done doing its work. This allows ASP.NET to do other work in the interim.

Asynchronous methods do not return a value immediately when they are called. They return a value at some later point in time. In other words you cannot access the return value immediately after calling an asynchronous method, you have to wait for it to finish.

The **await** keywords tells the current job to sleep until the asynchronous method returns a value. As such, using the **await** keyword makes the method it is used in asynchronous as well, since it can no longer return results immediately.

Asynchronous methods must be marked with the **async** keyword and they must return a **Task** or **Task<T>.** The **Task/Task<T>** object gives us a way to monitor the state of an asynchronous method and do other work in the interim if needed.

All of the methods in LINQ can be categorized as **deferred** or **non-deferred.**

- The **deferred** methods are used to construct a SQL Query (often a SELECT query) that will be run at some later time. (When working with collections these return iterators.)

    - Examples: Join(), Where(), OrderBy(), OrderByDescending(), Select()

- The **non-deferred** methods cause SQL Statements to be executed immediately. Calling the synchronous versions of these methods causes the current thread to be blocked and unavailable to handle other work until the database finishes its work. (These methods cannot return iterators.)

    - Examples: Count(), ToList(), ToArray(), SaveChanges(), First(), Last(), Single()

The synchronous LINQ methods are in the **System.Linq** namespace.
The asynchronous LINQ methods are in the **System.Data.Entity** namespace (which is the primary namespace for Entity Framework.)

Paging

Assuming that the first page is **1** and that each page contains **N** items.
(Remember that we index lists/arrays starting at index 0.)

- ❏ The first page will contain items **0** to **N-1**
- ❏ The second page will contain items **N** to **2N-1**
- ❏ The third page will contain items **2N** to **3N-1**
- ❏ etc.

Thus assuming we want to display the items for page **P.**

Page **P** will contain items **(P-1)*N** to **(P-1)*N - 1**

To implement this in LINQ we use the following code.

.Skip((page - 1) * pageSize).Take(pageSize)

Most tables / search results that we want to display to the user will have more rows than we can practically or efficiently show on one page, so we want the code to support the UI for pager to be as reusable as possible.

In fact there is no shortage of nuget packages that will happily provide their own paging interface.
There are currently over 200 nuget packages for paging.
https://www.nuget.org/packages?q=paging

Unfortunately many of these packages have not been built to support Bootstrap 4, prevent us from moving our search query and paging into a Stored Procedure, lack customization, and come with a lot of other unwanted dependencies. Building our own, gives us full access to change the code as needed and to stay up to date with changes in technology.

The PagingHelper class that we built is a custom HtmlHelper. Common uses for custom HtmlHelpers include:

- ❏ Paging
- ❏ Tabbed Navigation (think categories)
- ❏ Sorting and/or Filtering a table

Understanding how to build HtmlHelpers is not a core part of this course, especially since there are so many nuget packages which already provide the above functionality.

Partial Views are a much lighter weight tool and can often be used in place of a custom HtmlHelper.

Searching

Any searching or filtering we want to do, implies a change to the WHERE clause of our SELECT query.

An important thing to remember any time you are adding searching/filtering/sorting functionality is to add the user's inputs to the routeValues in the pager. Otherwise page 2 (and all pages after) will not contain the correct results.

So the basic process of implementing searching/filtering is such:

1. Add input parameters to the Index() method
2. Integrate those input parameters into the WHERE clause.
3. Pass all of the inputs along to the view.
4. Update the pager to include the inputs in the page links (via routeValues).
5. Finally, build the UI to do the filtering/searching.

All 5 steps are absolutely vital and I strongly recommend following them in the above order.

When a category filter or any filter that the user picks from a dropdown a simple == is sufficient and also efficient.

If users are entering text via a textbox, then it is preferable to use Contains() to search for the entered word anywhere in the text.

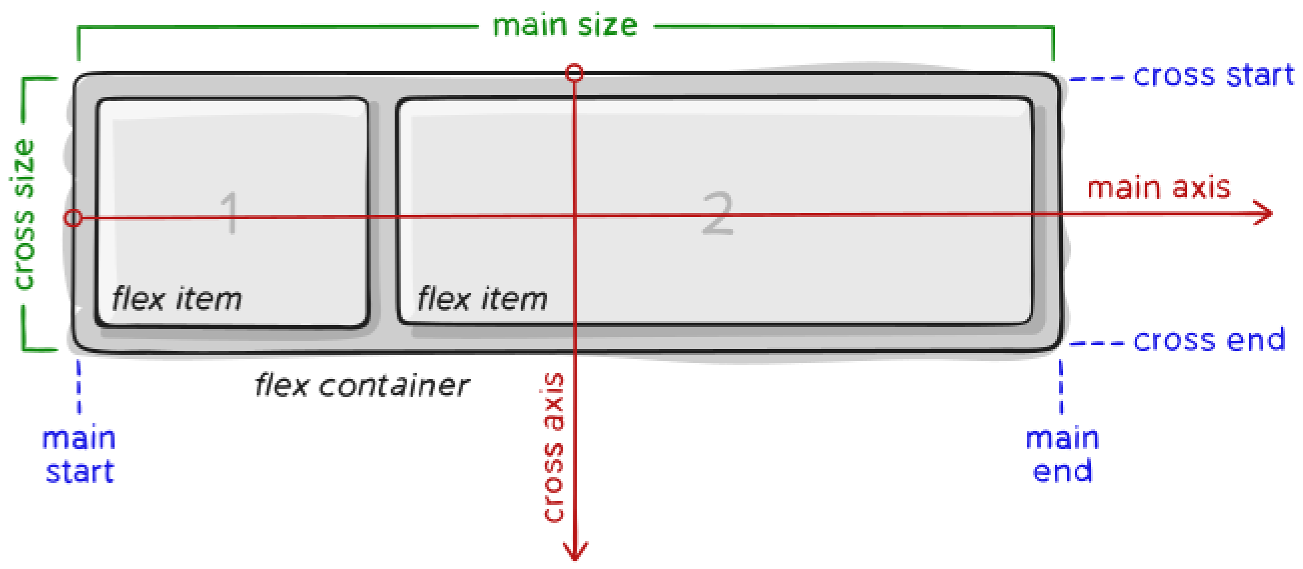
> *When searching for multiple keywords, a common technique is to Split() the input on spaces and then to add a Where() clause for each word.*

Filtering data for a particular date range, generally means:

- ❏ Let the user select a start date and an end date. *(The user does not have to enter both a start date or an end date, they may specify neither side, just one side, or both sides.)*
- ❏ If the user selects a start date, only show data for that date and later. (ex. posted >= startDate)
- ❏ If the user selects an end date, only show data for that date and earlier. (ex. posted <= endDate)

Flexbox

Flexbox is an extremely useful layout tool that was added to CSS in 2017. It is now a vital tool in creating layouts that scale well from smartphones up to desktop monitors and TVs.



Read the following article for more in depth information about flexbox.
https://css-tricks.com/snippets/css/a-guide-to-flexbox/

Play through the following game to review flexbox.
https://flexboxfroggy.com/

The Bootstrap 4 grid system is built on top of Flexbox, so understanding Flexbox is critical to tweaking the grid system.

Bootstrap 4 Grid System

The bootstrap grid system always has 12 columns. To use the grid system you must first add a <div class="row"></div> to contain the columns/cells. You can specify the width of an item in the grid by applying column classes to the direct children of the row.

col-12 the item is 100% width on extra-small screens and larger.
col-6 the item is 50% width on extra-small screens and larger.
col-3 the item is 25% width on extra-small screens and larger.

col-4 the item is ⅓ width on extra-small screens and larger.
col-2 the item is ⅙ width on extra-small screens and larger.
col-1 the item is 1/12 width on extra-small screens and larger.

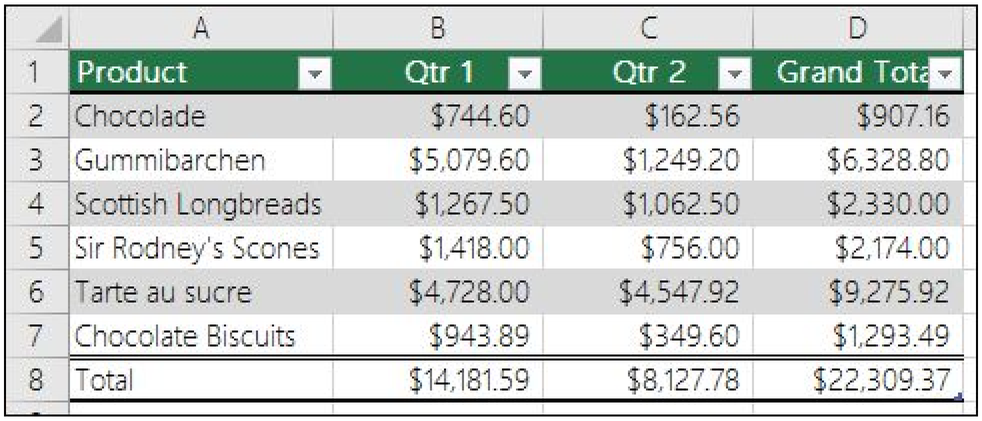
Use the col-sm-, col-md-, col-lg, and col-xl prefixes to change the number of columns on larger screens. Remember that Bootstrap is mobile-first, so you need to optimize the layout for smartphones first and then work up from there.

Read more about the grid system: https://getbootstrap.com/docs/4.0/layout/grid/

Showing Lists of Stuff

Depending on the specific industry and what kind of data you are showing. Below are three common options.

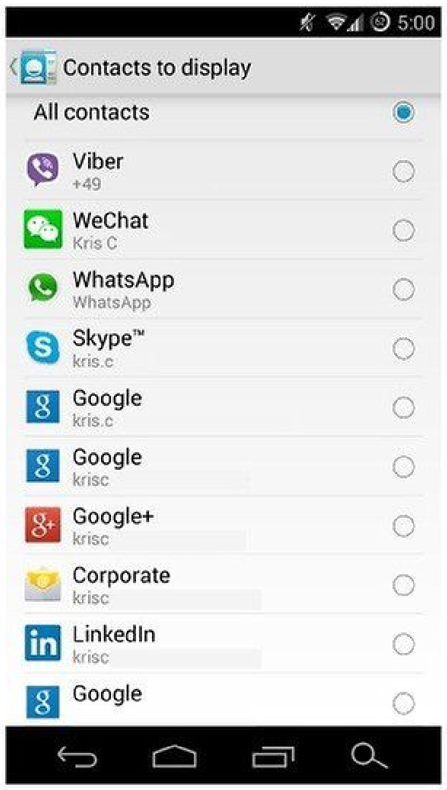
Table



	A	B	C	D
1	Product	Qtr 1	Qtr 2	Grand Tota
2	Chocolade	$744.60	$162.56	$907.16
3	Gummibarchen	$5,079.60	$1,249.20	$6,328.80
4	Scottish Longbreads	$1,267.50	$1,062.50	$2,330.00
5	Sir Rodney's Scones	$1,418.00	$756.00	$2,174.00
6	Tarte au sucre	$4,728.00	$4,547.92	$9,275.92
7	Chocolate Biscuits	$943.89	$349.60	$1,293.49
8	Total	$14,181.59	$8,127.78	$22,309.37

Looks like a spreadsheet, does not scale well.

List



5:00

Contacts to display

All contacts

Viber
+49

WeChat
Kris C

WhatsApp
WhatsApp

Skype™
kris.c

Google
kris.c

Google
krisc

Google+
krisc

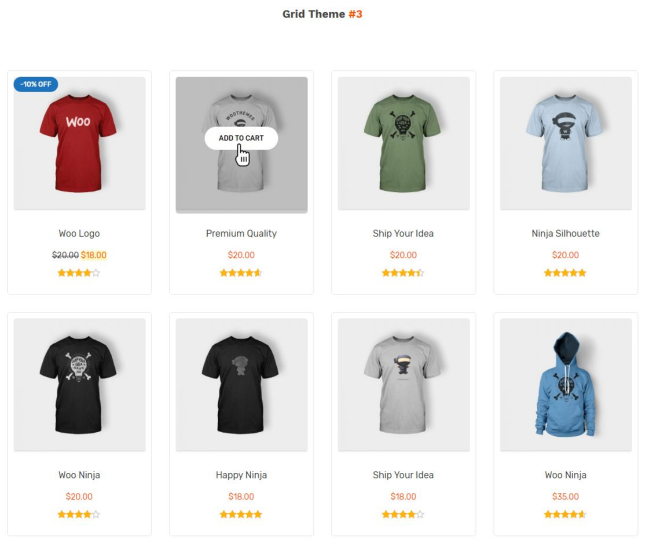
Corporate
krisc

LinkedIn
krisc

Google

Used primarily on smart phones.
Scales well to different screen sizes, but can waste space on larger screen sizes.
Recommend adding other UI blocks horizontally on large screens.

Grid



Used in ecommerce sites, image hosting sites, youtube, and games. Works well when each item has a high quality image. Scales the best to different screen sizes.

Handling Failure

Any time you interact with the database, something can go wrong and you may get an exception.

This can happen even if you are just executing a SELECT query.

But it happens most often when you are manipulating the data.

A good rule of thumb is to wrap any logic that manipulates the database with a try-catch clause. More specifically, if you are calling **SaveChanges** or **SaveChangesAsync**, then all of the code that is involved in inserting/updating/deleting rows in addition to the **SaveChanges** call should be wrapped in a try clause. And the catch clause should add information about the error to ModelState (if possible).

Remember to upload your code every day!

Minimum: 3 times / week
Preferred: 3 times / day

Items not covered or demonstrated yet

- ❏ Searching with multiple keywords (only demonstrated with one keyword)
- ❏ Using a Bootstrap 4 modal dialog to confirm that the user actually wants to delete something.
- ❏ Scaling down images on upload (to reduce file sizes, enforce aspect ratios, or create thumbnails)
- ❏ Saving uploaded files to disk.
- ❏ Validating that images are in an allowed format.
- ❏ Validating that image files are not too large.
- ❏ Using the <input type="file"> tag to specify valid file extensions.