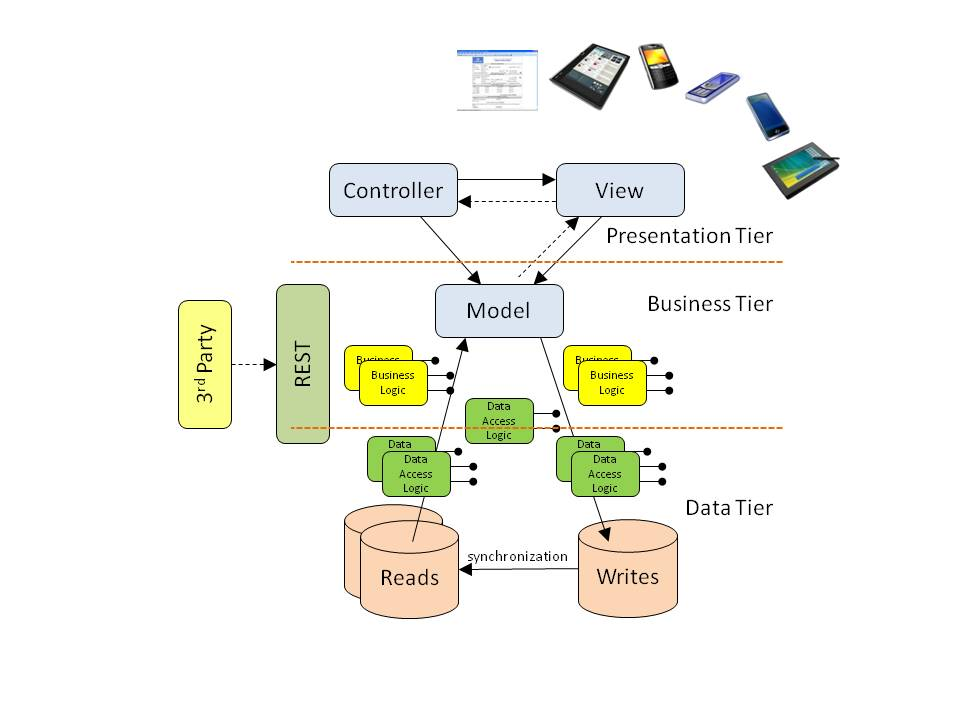
**Self-learning**

Entity frame work

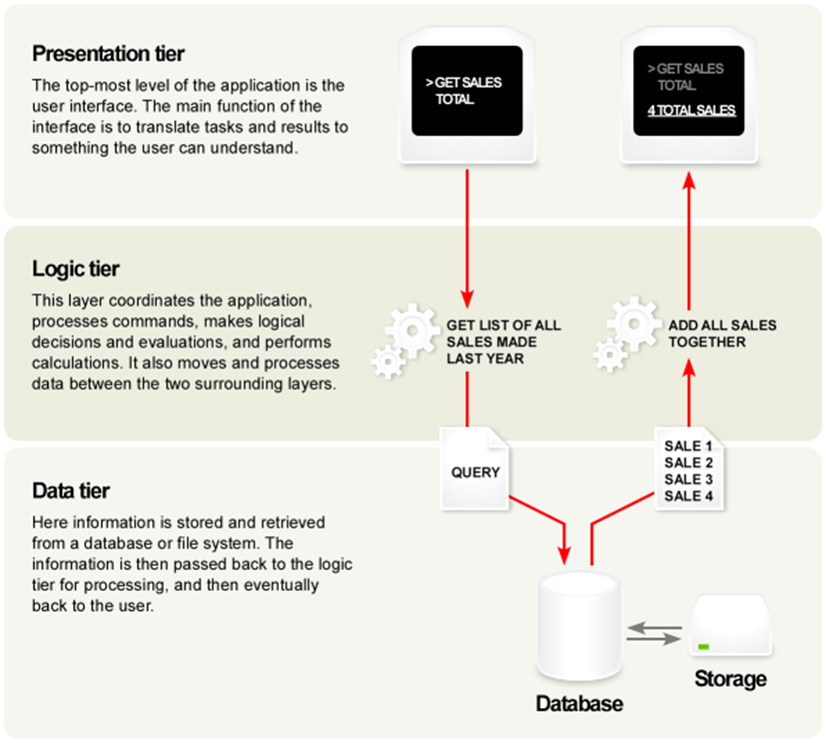
* Older architectures were, like modern enterprise architectures, built using a client server model. However unlike modern enterprise architectures the server side was not seperated into layers.
* The presentation, processing, data retrieval/update (and sometimes data storage) were all wrapped up into a single monolithic application/code module.
* This led to serious problems:
* It was not re-useable
  + As the processing, presentation and data access is tied together it was difficult to split it out and reuse parts of the code without heavy rewriting / restructuring.
* It did not handle scaling well.:
  + Increased data – As the data storage and access is tied in with the logic and presentation parts of the application you cannot move it onto a different machine to handle the extra data.
  + Increased load – You cannot split out the presentation, processing and data storage/access to run on separate machines so as to reduce the load on each.
  + Increased modules / processes – Creating new modules would require duplication of the data access and presentation code as these aspects were not seperate.

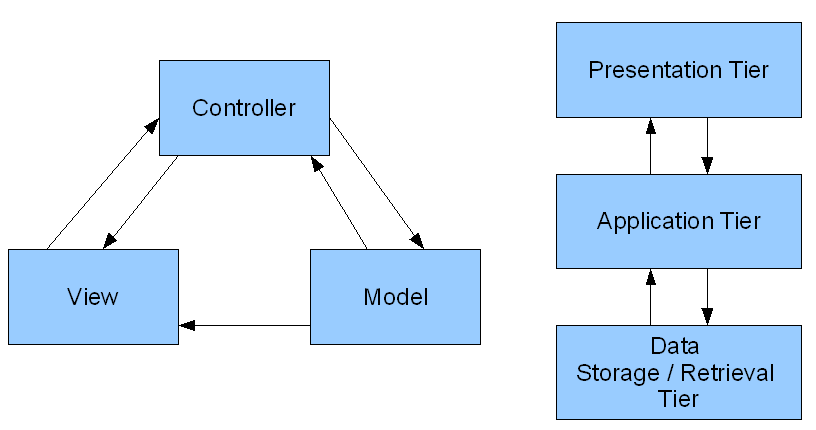
The Solution: N – tier architectures:

* The application is broken into separate tiers, each of which performs a different function



1. Tier (Presentation): Views (this is what the V stands for in MVC
2. Tier (Logic): Usually the Models and some Helperclasses (this is what the M stand for)
3. Tier (Data): With the help of the Entity Framework you could create a database from your models.





**J2EE**

* An architecture for implementing multi-tier enterprise applications.
* Standardises enterprise application architecture for java.
  + Standard system for handling http requests responses
  + Standard system for accessing system / server resources
  + Standard system for generating html / xml presentation
  + Standard system for database connectivity
* Portable: Built on Java SE. These standards remain the same across platforms / hardware
* Component / Container design philosophy
* JSP- Standard system for generating dynamic web content (Usually HTML).
* Much easier to use than servlets. (You can write plain html).
* No need to compile (The container compiles JSP pages into servlets when they are requested)
* Scriptlets
  + System for adding dynamic content to JSP pages.
  + Difficult to reuse.
  + Can become unmaintainable
* JSP tags
  + Ability to add dynamic content to JSP pages without embedding java code in the JSP itself
  + Java code is stored seperately from the JSP page
  + Tags are reuseable in different JSP pages
* JSTL
  + Standard set of tags for performing functionality common to most JSP pages
* Standard system for generating dynamic web content (Usually HTML).
* Much easier to use than servlets. (You can write plain html).
* No need to compile (The container compiles JSP pages into servlets when they are requested)
* Scriptlets
  + System for adding dynamic content to JSP pages.
  + Difficult to reuse.
  + Can become unmaintainable
* JSP tags
  + Ability to add dynamic content to JSP pages without embedding java code in the JSP itself
  + Java code is stored seperately from the JSP page
  + Tags are reuseable in different JSP pages
* JSTL
  + Standard set of tags for performing functionality common to most JSP pages

**Slide show**

The Carousel plugin is a component for cycling through the elements like a carousel (slideshow).

**carousel**  
Creates a carousel  
 **slide**Adds sliding animation effect when transitioning from one item to the other.  
 **carousel-inner**

This class is applied on the element, which contains all the slides of the carousel.

**item**Specifies the conent of each slide. Content can be text and images.  
  
**carousel-caption**Specifies a caption for the carousel.  
  
**carousel-indicators**Adds the dot indicators at the bottom of each slide, which indicates the current slide the user is viewing and the the total number of slides.  
  
**carousel-control**Adds the left or right buttons on the carousel to go back or forward one slide. To add left button, use left class along with carousel-control class and to add right button use right class along with carousel-control.  
  
**data-interval**Specifies the time delay in milli-seconds for transitioning from one slide to another. Set this attribute to false, if you do not want automatic sliding.  
  
**data-pause**The default value is hover. Pauses from transitioning to the next slide on hover. Set this attribute to false to stop the ability to pause on hover.  
  
**data-wrap**The default value is true, which means the carousel transitions through all the slides without stopping at the last slide. To stop at the last slide, set this attribute to false.  
 **data-ride="carousel"**It activates the carousel. The carousel can also be manually activated by using JavaScript.  
$("#carousel\_Id").carousel();  
  
**data-slide-to**Specifies which slide to go to when clicked. Slide index is ZERO based.  
 **data-slide**This attribute is added to the carousel controls (LEFT and Right buttons). For the left button, the value is "prev" and for the right button, the value is "next".

ISS

