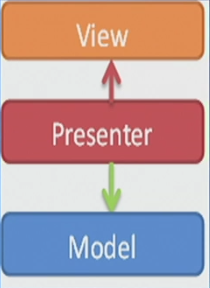
**Architectural Patterns for Model Driven Development**

**MVC, MVVM & MVP**

* The Problem
* View (UI)
* Model (Data Displayed in UI)
* Intermediary to enable view display data form the model that at changes as a result of user interaction – Glue code for Event handling, binding, business logic etc.

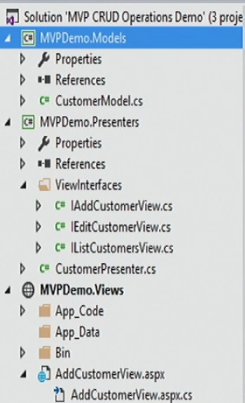
**MVP**



View relies on a Presenter to populate it with model data, react to user input, and provide input validation.

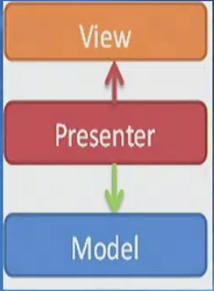
Handling is not in code behind, but now in presenter.

MVP Demo:



MVPDemo.Models is C# library (model)

MVPDemo.Prsenters is C# library (interface)



Model (Business rule, data access, model classes)

View (User interface (XAML))

ViewModel (middle man between view and model)

ViewModel

Interface between model and View. Provides data binding between View and model handles all UI actions using commands.

ViewModel

Does not need a reference to a view binds its control value to properties on a ViewModel.

One ViewModel per View

<TextBlock Text=”{Binding Name}”/>

**MVC and MVP**

* MVC – Controller action decided what view is displayed as app loads
* MVP where actions route through the View to the Presenter
* MVC, every action in View calls to a Controller with an action.
* MVP, view binds to the Model directly through data binding. Presenter’s job to pass off the Model to the View for binding; Presenter will also contain logic for gestures like pressing a button, navigation; Some code in code behind of view in order to delegate (register) to the presenter.
* MVC, a view does not directly bind to the Model. The view simply renders, and is completely stateless. No code behind; controller itself returns view while responding to URL action.

**MVC, MVP and MVVM**

* MVC – view know about the controller; the controller knows the model. The view is notified when the model changes.
* MVP – The presenter listens to the events of both the view and the model and mediates the interaction between both the view and model.
* MVVM – The view model exposes the data and command objects that the view needs. The view model pulls its data from the model(s).

**Where to Use what in the .NET World**

* Model-View-Controller(MVC) pattern
  + ASP.NET MVC
  + Disconnected Web Based Application
* Model-View-Presenter(MVP) pattern
  + Web Forms/SharePoint, Windows Forms
* Model-View-ViewModel(MVVM)
  + Universal Applications, WPF, Silverlight
  + Declarative Data binding

**MVC**

Model = what it is.

View = what it looks like.

Controller = what it does.



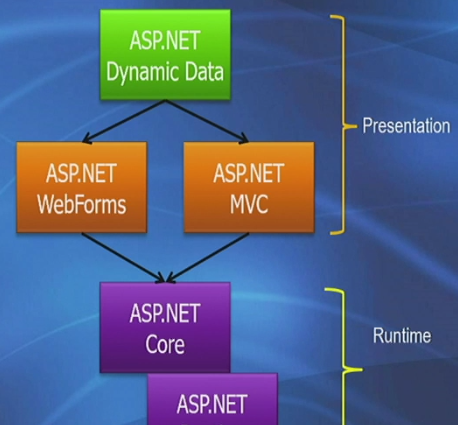
ASP.NET MVC

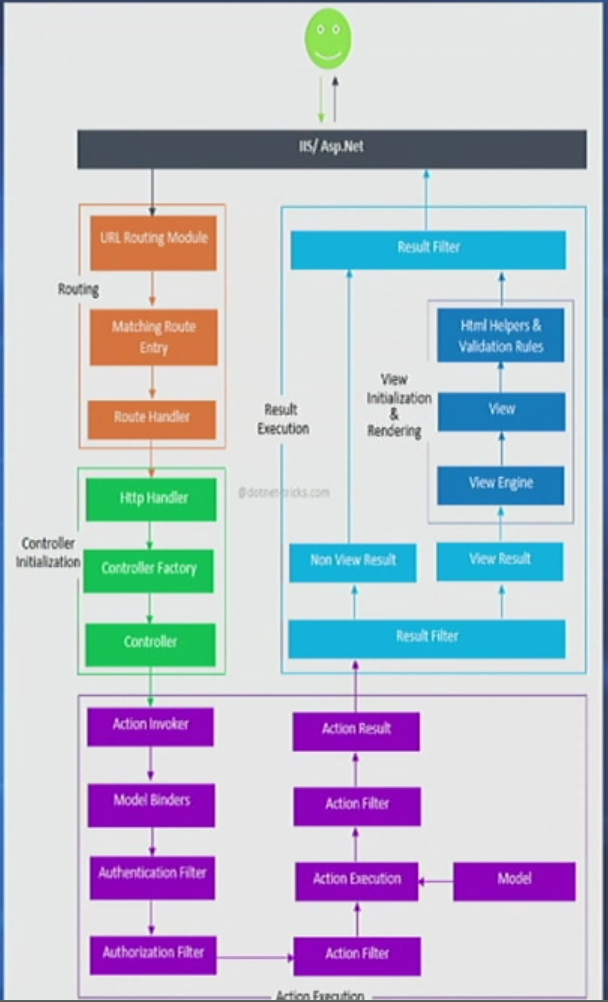
Past:



One web application framework to rule them all ….

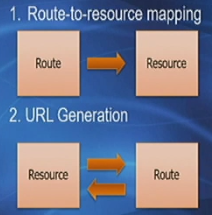
Now:







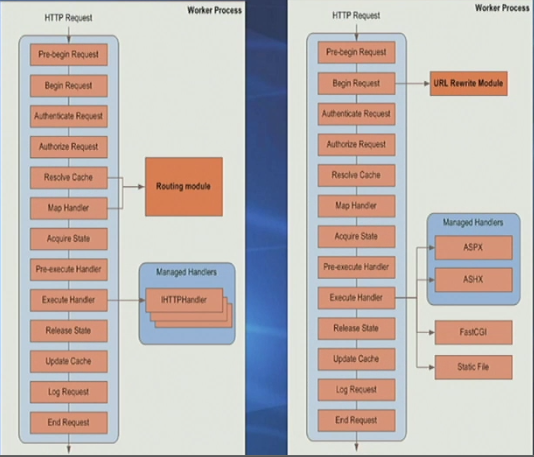
**What is ASP.NET Routing?**



**ASP.NET Routing**

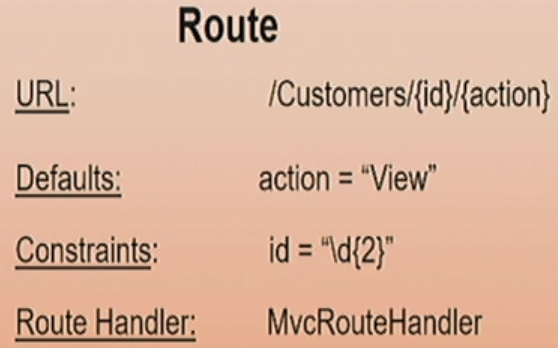
* Define your application’s URL entry points
  + Static (/Products/Edit/23)
  + Parameterized (/Products/{action}/{id})
* Map these URL patterns to route handlers
* Generate URLs based off your defined route table
* Can be leveraged by ASP.NET….
  + WebForms
  + MVC (using)
  + Dynamic Data (using)

**Workflow: Routing VS. Rewriting**



**Route**

* Resource map
  + URL (w/Parameters)
  + Constrains
  + Defaults
  + Route handler



**Pipeline**

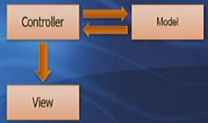


**Routing Benefits**

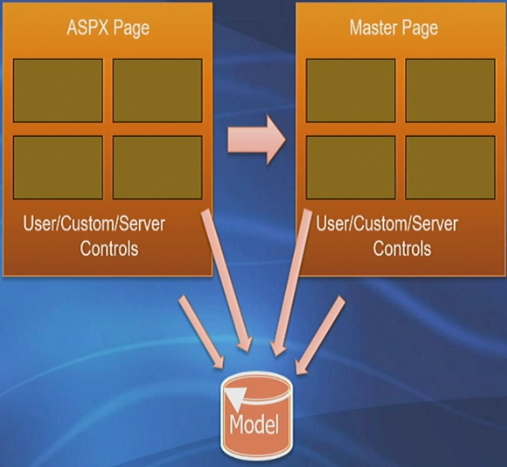
* Enables you to use URLs that do not have to map to specific in a Web site
* You can use URLs in a Web application that are descriptive of the user’s action
  + More easily understood by users
  + SEO Friendly
* You define URL patterns and placeholders
  + Apply business rules
  + Extend routing parser and handlers
* Programmatically create URLs that correspond to the routes
  + Centralized the logic for creating hyperlinks in your app

**What is MVC?**

* A design and architectural pattern
* Acronym for Model.View.Controller
* Separation of concerns

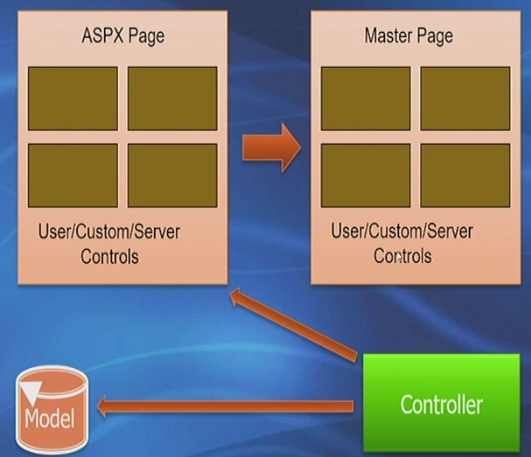


**Background: WebForms**

  
**What is ASP.NET MVC?**

* A new Web Application Project type
* Simply an option
  + Not a replacement for WebForms
  + Builds on top ASP.NET
  + Manual vs. automatic transmission

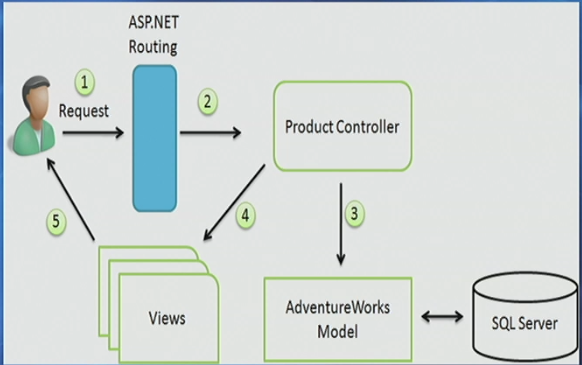
**ASP.NET MVC**



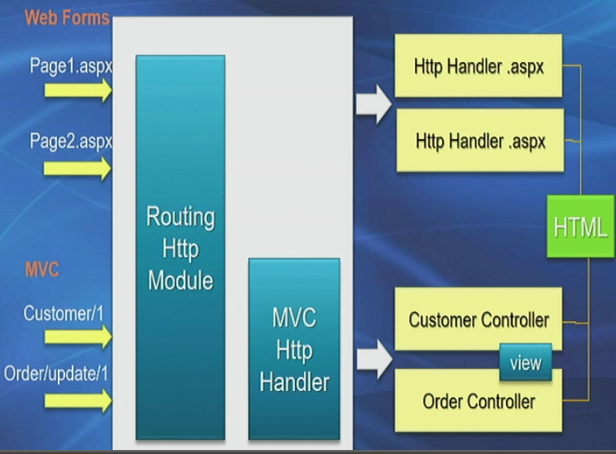
**Front Controller (MVC)**



**ASP.NET MVC: Application 101**



**ASP.NET MVC Tenets**



* Alternative – builds on ASP.NET, doesn’t replace it.
* Testable – WebForms are hard to test, MVC is designed for testing.
* Extensible – replace just about anything with you own implementation
* Routable – support search engines with easily discoverable URL patterns

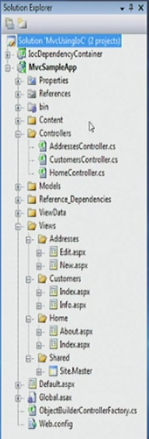
**ASP.NET MVC doesn’t have …**

* Postbacks
* View state
* Control state
* server-side form
* page/Control lifecycle

**ASP.NET MVC still has …**

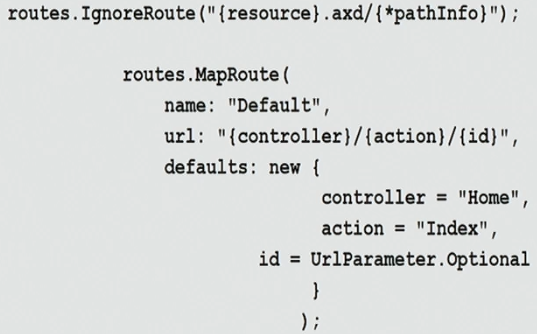
* Web designer
* Master pages
* User controls
* Membership/Roles/Profile
* Globalization
* Caching
* HTTP intrinsics:
  + HttpContext
  + HttpRequest
  + HttpResponse
  + Etc.

**Project Structure**

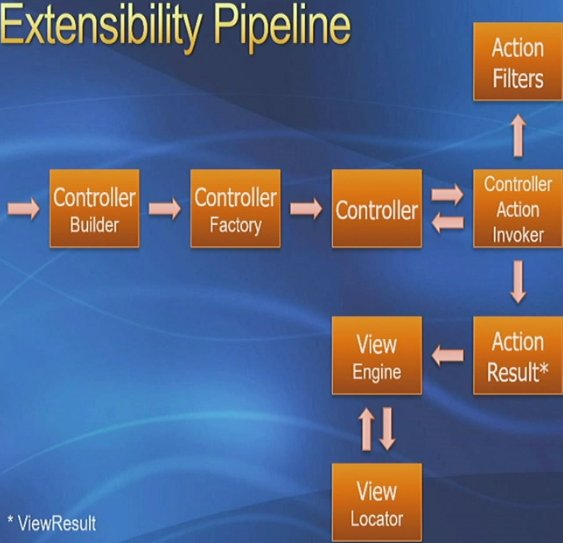


**Controller Conversions**

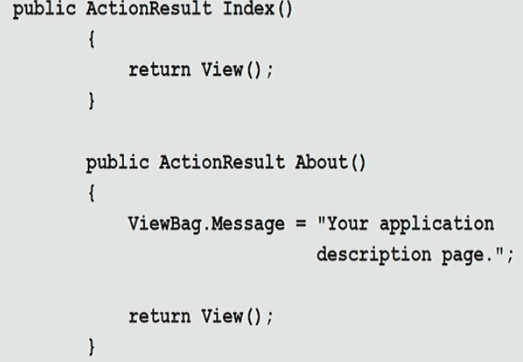
* Controller
  + Must …
    - Be suffixed with “Controller”
    - Implement IController (or inherit form Controller)
* Action
  + Must…
    - Be public
    - Return ActionResult or void
  + Can’t…
    - Be generic
    - Have a NonActionAttribute
    - Have out/ref parameters



**Extensibility Pipeline**



**Routing to Action**



**Action Results**

* ActionResult
* ContentResult
* EmptyResult
* JsonResult
* RedirectResult
* RedirectToRouteResult
* ViewResult

**Routing to Action**

Public ActionResult Index()

{

Return View();

}

public ActionResult About()

{

ViewBag.Message = "IIT MVC Application.";

return View();

}