What is ViewModel in MVC?

Ask Question

376

197

I am new to ASP.NET MVC. I have a problem in understanding the purpose of a ViewModel.

What is a ViewModel and why do we need a ViewModel for an ASP.NET MVC Application?

It is better if I can have a simple example.

asp.net-mvc viewmodel

edited Jul 24 '18 at 13:19

MikaS
7,364 14 21 34

asked Jun 16 '12 at 14:36

unique
2,147 4 18 24

4 This post is what you look for - "What is an ASP.NET MVC ViewModel?" – EL Yusubov Jun 17 '12 at 19:43

6 This article looks great:

possible duplicate of [In MVC, what is a ViewModel?](#) – rogerdeuce Jul 10 '15 at 18:24

12 Answers

547

A `view model` represents the data that you want to display on your view/page, whether it be used for static text or for input values (like textboxes and dropdown lists) that can be added to the database (or edited). It is something different than your `domain model` . It is a model for the view.

Let us say that you have an `Employee` class that represents your employee domain model and it contains the following properties (unique identifier, first name, last name and date created):

```
public class Employee
{
     public int Id { g

     public string Fir

     public string Las

     public DateTime D
}
```

View models differ from domain models

want to use on your view. For example, lets say that you want to add a new employee record, your view model might look like this:

```
public class CreateEmp
{
     public string Fir

     public string Las
}
```

As you can see it only contains two of the properties. These two properties are also in the employee domain model. Why is this you may ask? `Id` might not be set from the view, it might be auto generated by the Employee table. And `DateCreated` might also be set in the stored procedure or in the service layer of your application. So `Id` and `DateCreated` are not needed in the view model. You might want to display these two properties when you view an employee's details (an employee that has already been captured) as static text.

When loading the view/page, the create action method in your employee controller will create an instance of this view model,

```
public class EmployeeC
{
     private readonly

     public EmployeeCo
     {
          this.employe
     }

     public ActionResu
     {
          CreateEmploy

          return View(
     }

     public ActionResu
     {
          // Do what e
     }
}
```

Your view/page might look like this (assuming you are using `ASP.NET MVC` and the `Razor` view engine):

```
@model MyProject.Web.V

<table>
     <tr>
          <td><b>First
          <td>@Html.Te
              @Html.Va
          </td>
     </tr>
     <tr>
          <td><b>Last
          <td>@Html.Te
              @Html.Va
          </td>
     </tr>
</table>
```

Validation would thus be done only on `FirstName` and `LastName`. Using Fluent Validation you might have validation like this:

```
public class CreateEmp
AbstractValidator<Crea
```

```
                .NotEmp
                .WithMe
                .Length
                .WithMe

            RuleFor(m =>
                .NotEmp
                .WithMe
                .Length
                .WithMe
        }
}
```

And with Data Annotations it might look this:

```
public class CreateEmp
{
    [Display(Name = "F
    [Required(ErrorMes
    public string Firs

    [Display(Name = "L
    [Required(ErrorMes
    public string Last
}
```

The key thing to remember is that the view model only represents the data that you want to use, nothing else. You can imagine all the unnecessary code and validation if you have a domain model with 30 properties and you only want to update a single value. Given this scenario you would only have this one value/property in the view model and not all the properties that are in the domain object.

A view model might not only have data from one database

example above about adding a new employee record. Besides adding just the first and last names you might also want to add the department of the employee. This list of departments will come from your `Departments` table. So now you have data from the `Employees` and `Departments` tables in one view model. You will just then need to add the following two properties to your view model and populate it with data:

```
public int DepartmentI

public IEnumerable<Dep
```

When editing employee data (an employee that has already been added to the database) it wouldn't differ much from my example above. Create a view model, call it for example `EditEmployeeViewMode l` . Only have the data that you want to edit in this view model, like first name and last name. Edit the data and click the submit button. I wouldn't worry too much about the `Id` field because the `Id`

```
http://www.yourwebsite
```

Take this `Id` and pass it through to your repository layer, together with your first name and last name values.

When deleting a record, I normally follow the same path as with the edit view model. I would also have a URL, for example:

```
http://www.yourwebsite
```

When the view loads up for the first time I would get the employee's data from the database using the `Id` of 3. I would then just display static text on my view/page so that the user can see what employee is being deleted. When the user clicks the Delete button, I would just use the `Id` value of 3 and pass it to my repository layer. You only need the `Id` to delete a record from the table.

Another point, you don't really need a view model for every action. If it is simple data then it would be fine to only use `EmployeeViewModel`. If it is complex views/pages and they differ from each other

I hope this clears up any confusion that you had about view models and domain models.

edited May 6 '16 at 11:56

answered Jun 17 '12 at 20:21

Brendan Vogt
15.6k 27 120 210

5 @Kenny: Then show it :) What I was trying to say is lets say you have a domain model with 50 properties and your view only needs to display 5 then it is no use in sending all 50 properties just to display 5. – Brendan Vogt Jul 17 '13 at 5:44

4 @BrendanVogt – you did a good job explaining that, but I don't understand what the cost is of "sending all 50 properties". Other code has already created a Model object, with all 50 properties, and it doesn't seem worthwhile to maintain another class just to *not* send 45 properties – especially if you *might* want to send any one of those 45 properties in the future. – Kenny Evitt Jul 17 '13 at 13:01

5 @BrendanVogt – I think maybe

useful, particularly that a ViewModel (can) "... combine values from different database entities" [where I'm assuming that the phrase is just as true were "database entities" to be replaced with "Model objects"]. But still, what specific problems were ViewModels intended to address? Do you have any links? I couldn't find anything myself. [And I apologize if I seem to be picking on you!] – Kenny Evitt Jul 17 '13 at 13:09

6 +1 for mentioning fluent validation – MUG4N Oct 12 '13 at 17:37

3 I'm sorry for being critical but this answer is, unfortunately, incomplete. Defining a viewmodel as only what you need display on your page is like asking "What is a car?" and receiving an answer "Its not an airplane". Well thats true but not very helpful. The more correct definition of a VM is "Everything you need to render your page." If you read down to the bottom I have identified the components you need to build your VM's correctly and easily, in many cases leveraging

models. – Sam May 15 '15 at 16:43

126

View model is a class that represents the data model used in a specific view. We could use this class as a model for a login page:

```
public class LoginPage
{
    [Required(ErrorMes
username?")]
    [DisplayName("User
    public string User
    [Required(ErrorMes
    [DisplayName("Pass
    public string Pass
    [DisplayName("Stay
    public bool Rememb
}
```

Using this view model you can define the view (Razor view engine):

```
@model CamelTrap.Model

@using (Html.BeginForm
    @Html.EditorFor(m
    <input type="submi
}
```

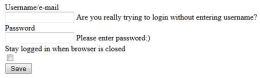
And actions:

```
[HttpGet]
public ActionResult Lo
{
    return View();
}

[HttpPost]
```

```
    return View(model);
}
```

Which produces this result (screen is taken after submitting form, with validation messages):



As you can see, a view model has many roles:

- View models documents a view by consisting only fields, that are represented in view.
- View models may contain specific validation rules using data annotations or IDataErrorInfo.
- View model defines how a view should look (for `LabelFor`, `EditorFor`, `DisplayFor` helpers).
- View models can combine values from different database entities.
- You can specify easily display templates for view models and reuse them in many places using DisplayFor or EditorFor helpers

retrieval: We want to display basic user data, his privileges and users name. We create a special view model, which contains only the required fields. We retrieve data from different entities from database, but the view is only aware of the view model class:

```
public class UserVM {
    public int ID { ge
    public string Firs
    public string Last
    public bool IsAdmi
    public string Moth
}
```

Retrieval:

```
var user = db.userRepo

var model = new UserVM
   ID = user.ID,
   FirstName = user.Fi
   LastName = user.Las
   IsAdministrator = u
   MothersName = user.
}
```

edited Aug 20 '14 at 19:09

VMai
8,882 6 16 31

answered Jun 16 '12 at 14:41

LukLed
26.5k 16 72 100

4 thank you,This is very helpful – unique Jun 16 '12 at 16:30

I thin user.Mother.FirstName + " " + user.Mother.LastName should be done

end. – Kurkula Jun 21 '15 at 16:38

3 @Chandana: I believe simple concatenation can be done in view model. There is no reason to expose two fields, if they are meant to be presented together. – LukLed Jun 22 '15 at 12:45

71

Edit: I updated this answer on my Blog:

http://www.samwheat.com/Post/The-function-of-ViewModels-in-MVC-web-development

My answer is a bit lengthy but I think it is important to compare view models to other types of commonly used models to understand why they are different and why they are necessary.

To summarize, and to directly answer the question that is asked:

Generally speaking, a view model is an object that contains all the properties and methods necessary to render a view. View model properties are often related to data objects such as customers and orders

application itself such as user name, application name etc. View models provide a convenient object to pass to a rendering engine to create a html page. One of many reasons to use a view model is that view models provide a way to unit test certain presentation tasks such as handling user input, validating data, retrieving data for display, etc.

Here is a comparison of Entity models (a.ka. DTO's a.ka. models), Presentation Models, and View Models.

Data Transfer Objects a.k.a "Model"

A Data Transfer Object (DTO) is a class with properties that match a table schema in a database. DTO's are named for their common usage for shuttling data to and from a data store. Characteristics of DTO's:

• Are business objects – their definition is dependent on application data.

• Usually contain properties only – no code.

• Properties exactly or closely match fields on a specific table in a data store.

Database tables are usually normalized therefore DTO's are usually normalized also. This makes them of limited use for presenting data. However, for certain simple data structures they often do quite well.

Here are two examples of what DTO's might look like:

```
public class Customer
{
    public int ID { ge
    public string Custo
}


public class Order
{
    public int ID { ge
    public int Customer
    public DateTime Or
    public Decimal Ord
}
```

Presentation Models

A presentation model is a *utility* class that is used to render data on a screen or report. Presentation models are typically used to model complex data structures that are composed from data from multiple DTO's. Presentation models often represent a

Characteristics of Presentation Models:

• Are business objects – their definition is dependent on application data.

• Contain mostly properties. Code is typically limited to formatting data or converting to or from a DTO. Presentation Models should not contain business logic.

• Often present a denormalized view of data. That is, they often combine properties from multiple DTO's.

• Often contain properties of a different base type than a DTO. For example dollar amounts may be represented as strings so they can contain commas and a currency symbol.

• Often defined by how they are used as well as their object characteristics. In other words, a simple DTO that is used as the backing model for rendering a grid is in fact also a presentation model in the context of that grid.

Presentation models

usually tied to the database schema). A presentation model may be used to model data for an entire page, a grid on a page, or a dropdown on a grid on a page. Presentation models often contain properties that are other presentation models. Presentation models are often constructed for a single-use purpose such as to render a specific grid on a single page.

An example presentation model:

```
public class Presentat
{
    public int OrderID
    public DateTime Or
    public string Pret
    public string Cust
    public Decimal Ord
    public string Pret
}
```

View Models

A view model is similar to a presentation model in that is a backing class for rendering a view. However it is very different from a Presentation Model or a DTO in how it is constructed. View models often contain the same properties as presentation models and DTO's and for this reason

Characteristics of View Models:

• Are the single source of data used to render a page or screen. Usually this means that a view model will expose every property that any control on the page will need to render itself correctly. Making the view model the single source of data for the view greatly improves its capability and value for unit testing.

• Are **composite objects** that contain properties that consist of application data as well as properties that are used by application code. This characteristic is crucial when designing the view model for reusability and is discussed in the examples below.

• Contain application code. View Models usually contain methods that are called during rendering and when the user is interacting with the page. This code typically relates to event handling, animation, visibility of controls, styling, etc.

• Contain code that calls business services for the

code is often mistakenly placed in a controller. Calling business services from a controller usually limits the usefulness of the view model for unit testing. To be clear, view models themselves should not contain business logic but should make calls to services which do contain business logic.

• Often contain properties which are other view models for other pages or screens.

• Are written “per page” or “per screen”. A unique View Model is typically written for every page or screen in an application.

• Usually derive from a base class since most pages and screens share common properties.

View Model Composition

As stated earlier, view models are composite objects in that they combine application properties and business data properties on a single object. Examples of commonly used application properties that are used on view

• Properties that are used to display application state such as error messages, user name, status, etc.

• Properties used to format, display, stylize, or animate controls.

• Properties used for data binding such as list objects and properties that hold intermediate data that is input by the user.

The following examples show why the composite nature of view models is important and how we can best construct a View Model that efficient and reusable.

Assume we are writing a web application. One of the requirements of the application design is that the page title, user name, and application name must be displayed on every page. If we want to create a page to display a presentation order object, we may modify the presentation model as follows:

```
public class Presentat
{
    public string Page
    public string User
    public string Appl
    public int OrderID
    public DateTime Or
```

```
    public string Pret
}
```

This design might work… but what if we want to create a page that will display a list of orders? The PageTitle, UserName, and ApplicationName properties will be repeated and become unwieldy to work with. Also, what if we want to define some page-level logic in the constructor of the class? We can no longer do that if we create an instance for every order that will be displayed.

Composition over inheritance

Here is a way we might re-factor the order presentation model such that it become a true view model and will be useful for displaying a single PresentationOrder object or a collection of PresentationOrder objects:

```
public class Presentat
{
    // Application prop
    public string Page
    public string User
    public string Appl

    // Business proper
    public Presentatio
}
```

```
    public string Appl

    // Business proper
    public List<Presen
}
```

Looking at the above two classes we can see that one way to think about a view model is that it is a presentation model that contains another presentation model as a property. The top level presentation model (i.e. view model) contains properties that are relevant to the page or application while presentation model (property) contains properties that are relevant to application data.

We can take our design a step further and create a base view model class that can be used not only for PresentationOrders, but for any other class as well:

```
public class BaseViewM
{
    // Application pro
    public string Page
    public string User
    public string Appl
}
```

Now we can simplify our PresentationOrderVM like this:

```
public class Presentat
```

```
public class Presentat
{
    // Business properi
    public List<Presen
}
```

We can make our BaseViewModel even more re-usable by making it generic:

```
public class BaseViewM
{
    // Application pro
    public string Page
    public string User
    public string Appl

    // Business properi
    public T BusinessOb
}
```

Now our implementations are effortless:

```
public class Presentat
{
    // done!
}

public class Presentat
{
    // done!
}
```

edited Oct 7 '16 at 15:15

answered Mar 19 '15 at 1:30

Sam
2,651 1 18 37

1 Sam Thank You!! this helped me fully grasp the multi-faceted entity that is a: View-Model. I'm a college student just learning the MVC architecture, and this clarified a bunch of

would put a star next to your answer. – Chef_Code Jan 26 '16 at 3:13

1 @Sam *'View models often contain the same properties as presentation models and DTO's and for this reason they are often confused one for the other.'* Does that mean they're commonly used **instead** of presentation models, or are they meant to contain the presentation models/dtos? – Alexander Derck Feb 1 '17 at 8:24

2 @AlexanderDerck They are used for different purposes. They are confused one for the other (in error). No, you typically will not use a pres model in place of a view model. Much more common is that the VM "contains" the presentation model i.e. `MyViewModel<MyPresModel>` – Sam Feb 1 '17 at 15:05

1 Way cool Sam! Thanks – Stokely Jul 13 '17 at 23:48

1 @Sam Assuming model objects are live objects e.g. nhibernate models.. so by having BusinessObject aren't we exposing model/live objects directly to the view? i.e. the business object can be used to modify the

require multiple business object properties, right? – Muhammad Ali Sep 30 '17 at 9:54

22

If you have properties specific to the view, and not related to the DB/Service/Data store, it is a good practice to use ViewModels. Say, you want to leave a checkbox selected based on a DB field (or two) but the DB field itself isn't a boolean. While it is possible to create these properties in the Model itself and keep it hidden from the binding to data, you may not want to clutter the Model depending on the amount of such fields and transactions.

If there are too few view-specific data and/or transformations, you can use the Model itself

answered Jun 16 '12 at 14:44

fozylet
1,061 1 12 23

I didn't read all the posts but every

that really helped me "get it"...

If a Model is akin to a database ***Table***, then a ViewModel is akin to a database ***View*** - A view typically either returns small amounts of data from one table, or, complex sets of data from multiple tables (joins).

I find myself using ViewModels to pass info into a view/form, and then transfering that data into a valid Model when the form posts back to the controller - also very handy for storing Lists(IEnumerable).

answered Sep 9 '16 at 20:53

halfacreSal
180 1 8

10

MVC doesn't have a viewmodel: it has a model, view and controller. A viewmodel is part of MVVM (Model-View-Viewmodel). MVVM is derived from the Presentation Model and is popularized in WPF. There should also be a model in MVVM, but most people miss the point of that pattern completely and they will only have a view

similar to the model in MVVM.

In MVC the process is split into 3 different responsibilities:

- View is responsible for presenting the data to the user
- A controller is responsible for the page flow
- A model is responsible for the business logic

MVC is not very suitable for web applications. It is a pattern introduced by Smalltalk for creating desktop applications. A web environment behaves completely different. It doesn't make much sense to copy a 40-year old concept from the desktop development and paste it into a web enviroment. However a lot of people think this is ok, because their application compiles and returns the correct values. That is, in my opinion, not enough to declare a certain design choice as ok.

An example of a model in a web application could be:

```
public class LoginMode
```

```
        {
            this.authentic
        }

        public bool Login(
        {
            return authent
        }

        public string User
        public string Pass
    }
```

The controller can use it like this:

```
    public class LoginCont
    {
        [HttpPost]
        public ActionResul
        {
            bool success =

            if (success)
            {
                return new
            }
            else
            {
                TempData["
                return new
            }
        }
    }
```

Your controller methods and your models will be small, easily testable and to the point.

edited Dec 3 '15 at 16:10

answered Dec 3 '15 at 16:04

Jeroen
726 5 21

Thank you for the insight into MVVM architecture, but why is MVC not OK? Your reasoning is

architecture such as MVC can mimic the behavior with-out having to write 50k lines of code, then whats the big deal? – Chef_Code Jan 26 '16 at 3:23

@Chef_Code: It is not questionable or favoritism: just read the original paper about MVC. Going back to the source is much better than blindly following the herd without question (aka "best practices"). MVC is meant for much smaller units: e.g. a button on a screen is composed of a model, view and controller. In Web-MVC the entire page has a controller, a model and a view. The model and view are supposed to be connected, so that changes in the model are **immediately** reflected in the view and vice versa. Mimicking is a very big deal. An architecture shouldn't lie to it's developers. – Jeroen Jan 26 '16 at 11:49

1 @jeroen The acronym MVC has been stolen and mangled. Yes MVC does not have a VM but it also doesn't have a Repository or a service layer and those objects are widely used in web sites. I believe the OP is asking "how do I introduce and

logic belongs. Business logic belongs in a service layer for a web or a desktop app using MVC or MVVM. The term model describes the business objects that are passed to/from the service layer. These definitions are vastly different from the original description of MVC. – Sam Feb 15 '16 at 16:46

1 @Sam Not everything that is part of a website, can be called part of MVC. There is no new meaning of MVC. There is the correct meaning and the "something completely unrelated that people confuse with MVC"-meaning. Saying that the model is responsible for the business logic, is not the same as business logic is coded in the model. Most of the time the model acts as a facade to the application. – Jeroen Feb 23 '16 at 13:48

The main flaw I see in Microsoft's MVC is the locking of a Model with a View. That itself defeats the whole purpose of all this separation that's been going on in N-Tier designs the past 20 years. They wasted our time forcing us to use "WebForms" in 2002 which was another Desktop-inspired

again another desktop model on this new paradigm for web dev. In the mean time Google and others are building giant client-side models that separate it all. Im thinking old ASP VBScript from 1998 was their truest web dev system. – Stokely Jul 13 '17 at 23:58

10

A lot of big examples, let me explain in clear and crispy way.

ViewModel = Model that is created to serve the view.

ASP.NET MVC view can't have more than one model so if we need to display properties from more than one models into the view, it is not possible. ViewModel serves this purpose.

View Model is a model class that can hold only those properties that is required for a view. It can also contains properties from more than one entities (tables) of the database. As the name suggests, this model is created specific to the View requirements.

- To list data from more than entities in a view page – we can create a View model and have properties of all the entities for which we want to list data. Join those database entities and set View model properties and return to the View to show data of different entities in one tabular form
- View model may define only specific fields of a single entity that is required for the View.

ViewModel can also be used to insert, update records into more than one entities however the main use of ViewModel is to display columns from multiple entities (model) into a single view.

The way of creating ViewModel is same as creating Model, the way of creating view for the Viewmodel is same as creating view for Model.

Here is a small example of List data using ViewModel.

Hope this will be

Sheo Narayan
1,061 2 12 15

10

View model a is simple class which can contain more than one class property. We use it to inherit all the required properties, e.g. I have two classes Student and Subject

```
Public class Student
{
public int Id {get; set
public string Name {get
}
Public class Subject
{
public int SubjectID {
public string SubjectNa
}
```

Now we want to display records student's Name and Subject's Name in View (In MVC), but it's not possible to add more than one classes like:

```
@model ProjectName.Mod
@model ProjectName.Mod
```

the code above will throw an error...

Now we create one class and can give it any name, but this format "XyzViewModel" will make it easier to understand. It is inheritance concept. Now we create a third class with the following name:

```
public string Name {ge
}
```

Now we use this ViewModel in View

> @model ProjectName.Model.StudentViewModel

Now we are able to access all the properties of StudentViewModel and inherited class in View.

edited May 31 '16 at 11:49

demonicdaron
399 3 17

answered Sep 19 '15 at 6:30

Mayank
121 1 4

6

ViewModel is workarround that patches the conceptual clumsiness of the MVC framework. It represents the 4th layer in the 3-layer Model-View-Controller architecture. when Model (domain model) is not appropriate, too big (bigger than 2-3 fields) for the View, we create smaller ViewModel to pass it to the View.

answered Oct 3 '17 at 9:03

1

A view model is a conceptual model of data. Its use is to for example either get a subset or combine data from different tables.

You might only want specific properties, so this allows you to only load those and not additional unneccesary properties

answered Sep 21 '18 at 9:16
user6685907

1

- ViewModel contain fields that are represented in the view (for LabelFor,EditorFor,DisplayFor helpers)
- ViewModel can have specific validation rules using data annotations or IDataErrorInfo.
- ViewModel can have multiple entities or objects from different data models or data source.

Designing ViewModel

```
public class UserLogin
```

```
 [Required(ErrorMessag
 [Display(Name = "Pass
 [MaxLength(50)]
 public string Passwor
}
```

Presenting the viewmodel in the view

```
@model MyModels.UserLo
@{
 ViewBag.Title = "User
 Layout = "~/Views/Sha
}
@using (Html.BeginForm
{
<div class="editor-lab
 @Html.LabelFor(m => m
</div>
<div class="editor-fie
 @Html.TextBoxFor(m =>
 @Html.ValidationMessa
</div>
<div class="editor-lab
 @Html.LabelFor(m => m
</div>
<div class="editor-fie
 @Html.PasswordFor(m =
 @Html.ValidationMessa
</div>
<p>
 <input type="submit"
</p>
</div>
}
```

Working with Action

```
public ActionResult Lo
{
return View();
}
[HttpPost]
public ActionResult Lo
{
// To acces data using
DataClassesDataContext
 if (ModelState.IsVali
{
try
 {
 var q = mobjentity.tb
user.Password).ToList(
 if (q.Count > 0)
 {
 return RedirectToActi
 }
```

```
catch (Exception ex)
{
}
}
return View(user);
}
```

1. In ViewModel put only those fields/data that you want to display on the view/page.
2. Since view reperesents the properties of the ViewModel, hence it is easy for rendering and maintenance.
3. Use a mapper when ViewModel become more complex.

answered Nov 10 '18 at 7:44

wild coder
400 8 13

0

View Model is class which we can use for rendering data on View. Suppose you have two entities Place and PlaceCategory and you want to access data from both entities using a single model then we use ViewModel.

```
public class Place
  {
     public int Plac
      public string
```

```
{
    public int ID
    public int? Pl
    public string
    public string
}
public class Place
{
    public string
    public string
    public string
    public string
}
```

So in above Example Place and Category are the two different entities and PlaceCategory viewmodel is ViewModel which we can use on View.

answered Jan 19 at 7:30

Sagar Shinde
81 8

protected by Samuel Liew ♦ Oct 5 '15 at 9:21

Thank you for your interest in this question. Because it has attracted low-quality or spam answers that had to be removed, posting an answer now requires 10 reputation on this site (the association bonus does not count).

Would you like to answer one of these unanswered questions instead?