

Entity vs Model vs View Model


asp.net-mvc

c#

entity-framework

mvvm

English (en) ▼



Question

I just spent some time reading about this terms (I don't use them that much since we don't have any MVC applications and I usually just say "model"), but I have the feeling these means different things depending on the context:

Entity

This is quite simple, it is one row in the database:

2) In relation to a database , an entity is a single person, place, or thing about which data can be stored.

Model

I often read, this is basically a combination of entities to represent a full set of data, let's say an Addresslist-model of a customer would combine the entities customer, address and probably individual.

Viewmodel

A term in the MVVM or MVC patterns, which is a model, which represents exactly the data you can see on the view. The viewmodel is on the application tier and has attributes for validation, f.e. [ASP.NET MVC Model vs ViewModel](#)

From my sight, these terms seem all a bit redundant: The Viewmodel has obviously his use, otherwise the view would have to do all the hard work to show the right stuff. The entity is just the representation, as we know from the EF, but if you combine these two, where has the model his use?

Stuff like validation, security etc. has to be done on the ViewModel. Would you use the model when you have hundreds of small tables to put another abstraction between the entities and the viewmodel? Or Are in terms of MVC and MVVM entities and models usually the same?

As usual thanks and a nice weekend

Matthias

[Matthias Mller](#)

Accepted Answer

Different people understand these terms a bit differently, but this is how I understand it:

Entity - object that has an identity (ID), usually comes from a database. Pretty simple class.

Model - any business object, this is a kinda broad term. It can be an entity, some custom class you've created in your project etc.. It's pretty much everything that isn't a view nor a controller/viewmodel.

ViewModel - some kind of a mediator between a model and the view. It modulates the communication between the model and the view, for instance applies validation, combines more models into one bigger object etc., for the purposes of the interaction with the specific view. ViewModel is also responsible for event handling (button mouse clicks for instance), so it exposes commands to the view you bind to (WPF).

[walther](#)

Popular Answer

The term "Model" is ambiguous. They are all models.

Entity Model

A class which closely resembles structure in persistence. A MemberEntity is a model which represents one member row in the Members table in a database. Not strictly tied to a Database, but some entity of some persistence. Typically has an "ID" property such as "int MemberID".

ViewModel

A class which closely resembles structure on a View/UI. A MemberViewModel is a model which represents one member to be displayed on a Members View/UI on the frontend of an application. Not strictly tied to the MV* pattern.

Notice

...that the above two models represent communication on the boundaries of the application. That is, the front boundary (entry point) which receives communication (user events and communication via protocol) to initiate business rules; And the

[Fastest Entity Framework Extensions](#) ➤

[+ Bulk Insert >](#)
[- Bulk Delete >](#)
[↻ Bulk Update >](#)

[🔗 Bulk Merge >](#)

back boundary which takes commands from business rules to open communication with other systems (such as databases or other endpoints).

Domain Model

A class which represents part of the problem domain. The MemberModel is responsible for its creation and validation. Because services take in and return out models, the models are responsible for their own business logic which validates their correct construction and usage. E.G.: A MemberModel should break if you try to use it without a UserName.

Domain Services

Domain Services take Entity Models and transform them into Domain Models so said services can work with the models. If an Entity comes in from the back boundary and fails to serialize or map into a Domain-Model, there is a red flag that the data is bad.

Domain Services take Domain Models and map them to Entities in order to send them out the back boundary. If the back boundary (DB/SDK?) fails to accept the model, the DB/SDK needs to be fixed.

- *Note: Entities conform to Models* because persistence is a detail. The domain is the king of a system, not the hardware or table-structure of persistence. The Domain is never wrong.

Front-Boundaries take ViewModels and transform them to Domain Models so they can be passed into the Domain. If a ViewModel fails to serialize or map into a Domain-Model, there is a red flag that the view/json/xml is bad.

Domain Services return Domain Models to the front boundary, which are then mapped to ViewModels in order to communicate out the front. If the View/UI fails to accept the model, the View needs to be fixed.

- *Note: ViewModels conform to Models* because consumers are a detail. The domain is the king of a system, not the UI's or Sub-Apps consuming them. The Domain is never wrong.

A ViewModel NEVER Knows about an Entity because a UI/Consumer never knows that persistence even exists.

Core Business-Logic should not know about ViewModels or Entities. Core Business-Logic only works with Domain Models. That's why Controllers, and Frontend-Services near them, exist; To map Domain Models \leftrightarrow ViewModels. That's also why SDK's, and Backend-Services near them, exist; To map DomainModels \leftrightarrow Entities.

When a system is built, the Domain and Business Logic are built first (Hopefully TDD). Then adapters are put on the Front and Back of the business logic which determine the Delivery-Mechanism (frontend) and the Dependencies (Service/Persistence) (Backend). But those frontends and backends could be ripped out, and the core business logic still exists.

Shorter Version (TLDR;):

Entity: Database Record.

Domain Model: Model-specific business logic (Google "Value Object") to represent an object in the Domain Problem.

ViewModel: Page (or section) of a View.

[Suamere](#)

 View more on Stack Overflow

Licensed under: [CC-BY-SA](#) with [attribution](#)
Not affiliated with [Stack Overflow](#)