MVC

**M**odel **V**iew **C**ontroller

Auch bekannt unter **MVVM** (**M**odel **V**iew **V**iew**M**odel) oder **MVP** (**M**odel **V**iew **P**resenter)

Diese Designpattern trennt Business Logik sauber von der Präsentationslogik. Durch diese saubere Trennung können Änderungen leicht vorgenommen werden.

Diese Trennung wird in **drei Klassen** aufgeteilt:

* **View**
  + Beschäftigt sich mit der Struktur und Darstellung der Daten, dem User Interface
  + Interagiert mit dem Controller mit DataBinding und Commands
* **Controller**
  + Koordiniert View Interaktionen mit dem Model
* **Model** 
  + Enthält Business Data und Logik
  + Implementiert bestimmte Interfaces um Benachrichtigungen bekannt zugeben



# View:

Die View ist ein Visuelles Element, könnte z.B. eine Page oder ein Window sein. Die View referenziert den Controller (z.B. mit DataContext in WPF) und ***bindet*** dann an bestimmte **Properties** oder **Commands** vom Controller.

Die View kann Änderungen zum Darstellen von den Properties selbst vornehmen, z.B. könnte die View ein **DateTime** in ein bestimmtes Ausgabeformat bringen.

Ebenfalls enthält die View ein Code-Behind File, wo bestimmte Darstellungen ausprogrammiert werden können, welche schwierig z.B. in XAML verwirklicht werden kann.

# Controller

Der Controller ist kein visuelles Element, sondern ist zuständig für einen spezifischen Task in der Anwendung. Der Controller enthält keine Instanz von der View, sondern nur vom Model, jedoch kann die View vom Controller mittels Events benachrichtigt werden (z.B. in C# INotifyPropertyChanged).

Ebenso hat der Controller Properties oder Commands, die für die View durch DataBinding verwendet werden können.

Der Controller leitet Interaktionen von der View an das Model weiter:

* Der Controller kann dabei die Daten in ein passendes Format konvertieren, falls sie nicht für das Model passen
* Somit kann der Controller auch durch Interfaces Fehlermeldungen an die View weiterleiten (z.B. IDataErrorInfo or INotifyDataErrorInfo)

# Model

Das Model ist ebenfalls kein visuelles Element. Es ist zuständig für die Business Logik der Anwendung, mittels Berechnungen oder Zugriffe an Datenquellen mittels DAL (Data Access Layer) an Datenbanken oder Rest APIs.

Das Model enthält **keine Referenzen** an den Controller oder die View, jedoch werden beide mittels Events benachrichtigt (z.B. **INotifyPropertyChanged** oder **INotifyCollectionChanged**).