MVC Architektur Model-View-Controller

Einst für die Programmiersprache „Smalltalk-76“ in den späten 70ern vom Norweger Trygve Reenskaug entwickelt, ist die MVC-Architektur heute eine der verbreitetsten Möglichkeiten, den Code einer Applikation zu strukturieren. Die MVC-Struktur besteht aus drei Komponenten: Model, Controller und View.

Model:

Das Model beinhaltet die eigentliche Logik des Programms. Neben dem Ausführen von Methoden, greift diese Komponente auf Datenquellen wie Datenbanken und APIs zu. Die dort gespeicherten Informationen werden vom Model verarbeitet und aktualisiert.

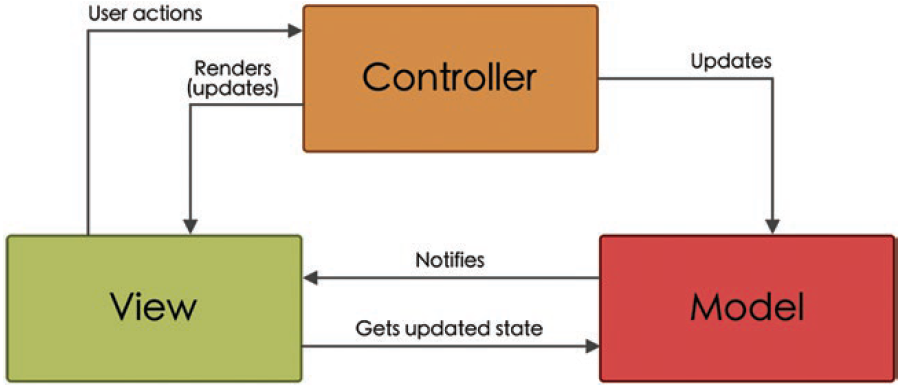
View:

In der View-Klasse sind alle Methoden zum Aufbau einer grafischen Oberfläche implementiert. In ihr ist das Layout der Anzeigeelemente wie Buttons und Listen festgelegt. Außerdem werden in Ihr Usereingaben entgegengenommen und an den Controller weitergeleitet. Sie dient der Visualisierung der von Model aufbereiteten Daten. In der modernen MVC-Architektur kommuniziert die View-Klasse nicht mehr direkt mit dem Model.

Controller

Der Controller bildet das Bindeglied der MVC-Architektur. Er leitet die Useranfragen und eingaben an das Model. Meldet die Model-Klasse das Daten aktualisiert wurden, übergibt er den Befehl zur Aktualisierung der Anzeige an View. Außerdem wird im Controller die App gestartet, hier werden die Initialisierungen durchgeführt.

In der ursprünglichen Version kommunizierten diese Komponenten untereinander.



In der heutigen Programmentwicklung wird versucht, Programme und Klassen möglichst modular anzulegen. Aus diesem Grund wurde die MVC-Architektur noch einmal optimiert. In der so genannten Apple MVC findet keine direkte Kommunikation zwischen Model und View mehr statt. Das macht die Programmstruktur noch übersichtlicher und ermöglicht es die einzelnen Klassen für sich zu testen, zu duplizieren und in anderen Projekten wieder zu verwenden. Das spart nicht nur Zeit, sondern auch Kosten.

Ein Bild, das Text, Screenshot, Rechteck, Klebezettel enthält.

KI-generierte Inhalte können fehlerhaft sein.

Ein weiterer Vorteil dieser Struktur ist ihre Einfachheit, sie ist auch für Programmeinsteiger gut verständlich und erleichtert spätere Anpassungen. Durch die klare Einteilung fällt die Aufgabenzuweisung leicht, was zu einem übersichtlichen und *wartbaren* Code führt. Außerdem reduziert sie das Risiko von Fehlern erheblich.

Bei der Implementierung ist es jedoch entscheidend, dieses Schema konsequent einzuhalten. Andernfalls können die Vorteile dieser Architektur nicht zum Tragen kommen. Besonders häufig tritt das Problem auf, dass in der MVC-Architektur, Controller und View sehr eng miteinander verflochten werden. In einer modularen Lösung sollte die Kopplung jedoch stets möglichst lose gehalten werden. Eine enge Verknüpfung erschwert nicht nur die Wiederverwendbarkeit der Komponenten, sondern auch das Debugging erheblich.

Wir haben uns für die MVC-Struktur entschieden, da sie auf uns sehr intuitiv wirkt und es die Zusammenarbeit von mehreren Entwicklern an einem gemeinsamen Code erheblich erleichtert. Durch die klare Trennung der Komponenten konnten wir die Aufgaben unter uns gut aufteilen. Zudem ermöglicht uns die Architektur, die Applikation zu einem späteren Zeitpunkt zu erweitern – wie im Kapitel *„Aussichten“* genauer beschrieben. Mit der MVC-Architektur schaffen wir eine gute Grundlage, auf welche wir unser Programm weiter aufbauen können.

Vorteile:

Gute für den Einstieg in die Programm Architektur.-> simpel

Wenigere code benötigt als in anderen Architekturen -> übersichtlicher

Klare Trennung der aufgaben Bereiche, da jede Komponente ihre eigene Zuständigkeit hat.

Sehr weit verbreitet, um nicht zu sagen „gängigste“ Architektur in Neuentwicklungen von Mobile App.

Nachteile:

Es geschieht schnell, dass in der MVC-Architektur, Controller und View sehr eng miteinander verwoben sind. Das untergräbt ein wichtiges Ziel der MVC-Architektur, die klare Trennung der Zuständigkeiten der modularen Komponenten. Dies erschwert die Wiederverwendung vor allem der Controller Klasse, wenn zum Beispiel Views nicht mehr online, sondern auf dem Desktop anzeigen soll. Muss Oft auch Controller modifiziert werden. In einer Modularen Lösung sollte die Kopplung immer möglichst lose gehalten werden. Außerdem erschwert es das Debuggen, wenn man nicht mehr genau unterscheiden kann, wo der Fehler geschieht.

„Massive View Controller“: Dadurch dass dem Controller zu viele Aufgabe zugeteilt werden, besteht die Gefahr der Überladung. -> Dem Controller werden Aufgaben zugeteilt, die nicht in seinen Bereich fallen Macht code unübersichtlich, schwerer zu überarbeiten

* Jede Komponente solle nur eine Aufgabe haben: Anzeigen-Steuern-Verarbeiten