

RUHR-UNIVERSITÄT BOCHUM

WEB-ENGINEERING

Sommersemester 2023



Informatik
im Bauwesen

Webengineering

**Model View ViewModel
(MVVM)**

Model View ViewModel

Beim Model-View-ViewModel (MVVM) handelt es sich um ein Entwurfsmuster dass die Geschäftslogik von der Benutzeroberfläche trennt

- Das MVVM Muster ist eine Variante des Model-View-Controller (MVC) Entwurfsmuster
- Das MVVM Muster nutzt die funktionale Trennung des MVC Musters zur Trennung von Model und View
- Das MVVM Muster wurde 2005 von John Gossman veröffentlicht

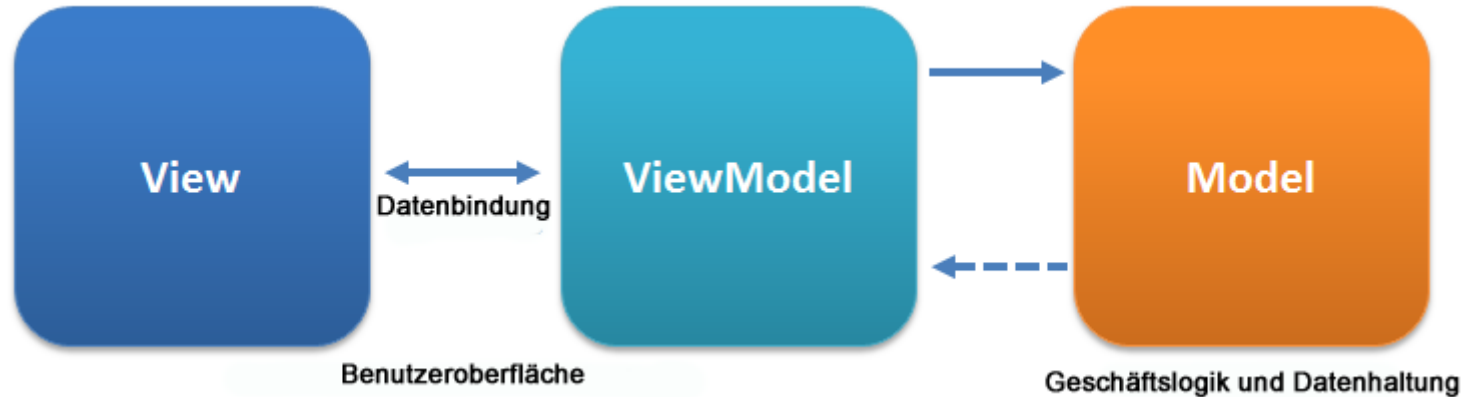
Model View ViewModel

Das MVVM-Muster besteht aus folgenden drei Komponenten:

- **View:** Beinhaltet die grafische Benutzeroberfläche (GUI). Innerhalb des Views können zusätzlich zu den grafischen Elementen, Verbindungen zur Verarbeitung von Nutzereingaben (z.B. Buttons) an das ViewModel erstellt werden. Der View implementiert keine eigene Geschäftslogik.
- **Model:** Verantwortlich für die Datenhaltung und Geschäftslogik. Innerhalb des Models befinden sich die Daten die dem Benutzer angezeigt werden. Weiterhin werden im Model mögliche Datenänderungen vorgenommen. Zuletzt validiert das Model vom Nutzer übergebene Daten.
- **ViewModel:** Implementiert die Eigenschaften und Funktionen des Views. Das ViewModel erfüllt die Aufgabe des Bindeglieds zwischen View und Model.

Model View ViewModel

Durch einen Datenbindungsmechanismus zwischen den drei getrennten Komponenten kann eine lose Kopplung erreicht werden



Model View ViewModel

Datenbindung zwischen View und ViewModel

- View bezieht Funktion und Logik vom ViewModel
- View wird über eine Datenbindung (Binder) mit dem ViewModel verbunden
- Die Datenbindung ist deklarativ und muss folgende Aufgaben erfüllen:
 - Verbinden der GUI-Elemente eines Views mit dem ViewModel
 - Beobachtung von Veränderungen im ViewModel und das Aktualisieren des Views
 - Beobachtung von Veränderungen im View und das Aktualisieren des ViewModels

Model View ViewModel

Durch die Trennung der Komponenten und anschließende Kopplung entstehen mehrere Vorteile:

- Da für die Erstellung des Views keine Programmierkenntnisse erforderlich sind, können diese von reinen Benutzeroberflächen Experten erstellt werden, während Entwickler unabhängig von der GUI das ViewModel und Model implementieren können.
- Durch die lose Kopplung, können die einzelnen Komponenten unabhängig voneinander einfacher gewartet und getestet werden.
- Unterschiedliche Views können das gleiche ViewModel einbinden, ohne das eine Änderung am ViewModel vorgenommen werden muss.
- Analog kann die Geschäftslogik bearbeitet werden, ohne den View bearbeiten zu müssen.

Frameworks

Als Framework wird ein Programmiergerüst in der komponentenbasierten Entwicklung bezeichnet.

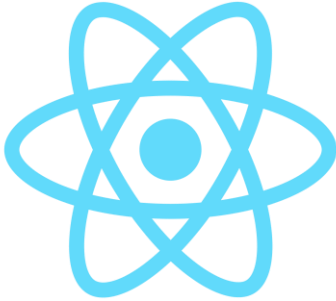
- Ist kein fertiges Programm, sondern stellt einen Rahmen für die Erstellung von Anwendungen zur Verfügung
 - Ein Toolkit enthält Funktionen und Hilfsklassen, ein Framework stellt zusätzlich eine kohärente Struktur zur Verfügung
- Ein Framework wird als `Black-Box` bezeichnet, wenn der Benutzer keine Details hinter der Schnittstelle und ihrer Spezifikation kennt
- Eine `White-Box` kann angepasst und erweitert werden, auch wenn Teile des Codes eine Kapselung erzwingen und begrenzen was der Benutzer tun kann



Quelle: <https://upload.wikimedia.org/wikipedia/commons/4/44/Blackbox3D.png>

JavaScript Frameworks

React



<https://reactjs.org/>

Angular



<https://angular.io/>

Vue.js



<https://vuejs.org/>

- React ist eine JavaScript-Bibliothek
- Angular & Vue.js sind JavaScript-frameworks
- Alle drei Technologien können zum Bau von User Interfaces genutzt werden

Webengineering

Vue.js Einführung

Vue.js

Vue.js ...

- erschien 2014 unter der Leitung von Evan You
- ist ein lightweight clientseitiges JavaScript Webframework zum Erstellen von Benutzeroberflächen
- ist eine MIT-lizenziertes Open Source Projekt und kostenfrei
- erstellt Single-Page-Webanwendung nach dem MVVM-Muster
- kann auch in Multi-Page-Webanwendungen für einzelne Abschnitte genutzt werden
- verbindet Elemente von React und Angular



Quelle: <https://vuejs.org>

Vue.js

Installation

Vue.js kann über mehrere Wege in ein Projekt eingebunden werden

- Einbinden von Vue über das script-Tag
 - Hierfür muss Vue.js vorher heruntergeladen werden
 - Nach dem einbinden über das script-Tag wird Vue als gloable Variable registriert

```
<script src='./vue.js'></script>
```

Vue.js

Installation

- Über Content Delivery Network (CDN)
 - Zum Lernen und Prototyping kann die neuste Version verwendet werden
`<script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>`
 - Für veröffentlichte Webseiten, sollte die genutzte Vue Version angegeben werden, um mögliche Fehler durch neuere Versionen vorzubeugen
`<script src="https://cdn.jsdelivr.net/npm/vue@2.6.11"></script>`
- Über Node Package Manager (NPM)
 - Bei größeren Projekten sollte die Installation über NPM bevorzugt werden

Kommandozeile

```
C:\Users\...> npm install vue
```

Vue.js Instances

- Jede Vue Anwendung (Vue-App) beginnt mit dem erstellen einer Vue Instance über die Vue()-Funktion

```
var vm = new Vue({  
  // options  
});
```

- Da Vue durch das Prinzip des MVVM Musters inspiriert ist, wird die Vue Instance per Konvention vm (kurz für ViewModel) benannt
- Beim Erstellen einer Vue Instance wird der Vue()-Funktion ein options Objekt übergeben
- Über die Attribute des options Objekts werden die Daten und das Verhalten der Vue Instance festgelegt

Vue.js Instances

- Bei der Erstellung einer Vue Instance werden alle Eigenschaften des data Attributes zu Vues reaktiven System hinzugefügt
- Wenn Eigenschaften innerhalb des reaktiven Systems geändert werden, werden diese direkt auf der Webseite aktualisiert

```
var vm = new Vue({  
  data: {  
    message: "Vorlesung Web-Engineering"  
  }  
});
```

- Bei einer Änderung der Eigenschaft message, wird die Webseite mit dem geänderten Wert von message neu gerendert

Vue.js Instances

Das data Attribut des options Objekts:

- Eigenschaften die **nach** der Erstellung der Vue Instance erstellt werden, werden nicht verbunden, d.h.:

```
vm.message2 = "Hallo Welt!"
```

→ Veränderung wird keine Aktualisierung der Webseite auslösen

- Sollten Eigenschaften erst später benötigt werden, sollten diese leer initialisiert werden

```
var vm = new Vue({  
  data:{  
    message: "Vorlesung Web-Engineering",  
    message2: "",  
    count: 0  
  }  
});
```


Vue.js Instances

Das methods Attribut des options Objekts:

- Funktionen innerhalb des methods Attributes können direkt über die Vue Instance aufgerufen werden.
- Der Kontext von this wird automatisch an die Vue Instance gebunden

```
var vm = new Vue({  
  data:{  
    count: 0  
  },  
  methods:{  
    add: function() {this.count++;}  
  }  
});  
vm.add()  
vm.count //ausgabe = 1
```

Achtung:

Innerhalb des methods Attributes sollten keine Arrow-Functions genutzt werden, da this hierbei nicht an die Vue Instance gebunden wird

Vue.js Instances

Das el Attribut des options Objekts:

- Das el Attribut definiert ein DOM-Element, auf welches die Vue Instance „aufgesetzt“ (eng.: mounted) wird
- Der Wert des Attributes kann hierbei ein CSS-String oder das eigentliche HTML-Element sein

```
var vm = new Vue({  
  el: "#app",  
  data:{  
    message: "Vorlesung Web-Engineering"  
  }  
});
```

- Sollte el bei Initialisierung der Vue Instance nicht gesetzt sein, muss dieses nachträglich gesetzt werden, da die Webseite ansonsten nicht automatisch von der Vue Instance zusammengestellt wird

Vue.js Instances

Vue Instances besitzen zusätzlich eine Reihe an eigenen Attributen und Funktionen

Diese werden mit dem Prefix **\$** von Benutzer erstellten Attributen und Funktionen unterschieden, z.B.:

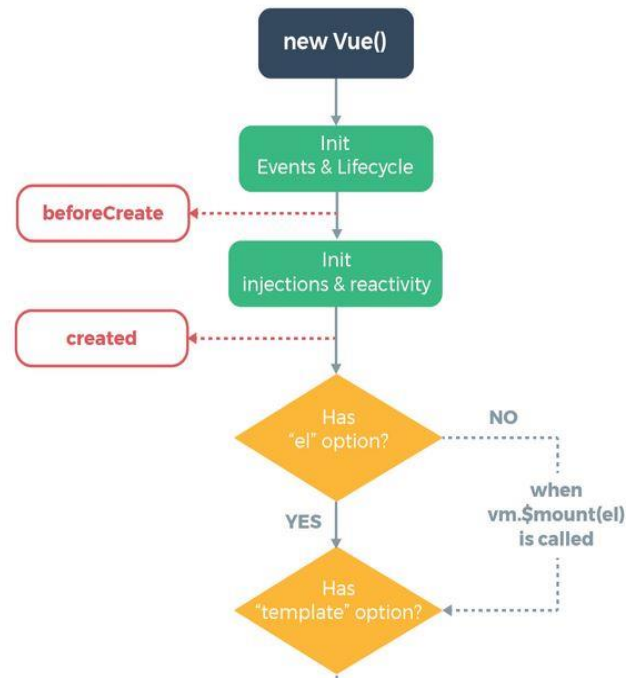
```
vm.$data // Gibt das Data Attribute der Instance zurück  
vm.$el // Gibt das DOM-Element der Instance zurück  
vm.$mount() // Übergibt der Vue Instance ein DOM-Element  
vm.$watch("a", function (newValue, oldValue) {  
    //Wird ausgeführt sobald sich die Eigenschaft a ändert  
})
```

Vue.js Instances - Lifecycles

Eine Vue Instance durchläuft mehrere Initialisierungsschritte bevor sie erstellt wird

- Vue Instances erlauben über Lifecycle Hooks den Zugriff auf diese Schritte
- Hooks können dem options Objekt hinzugefügt werden, um Benutzer-spezifischen Code an diesem Schritt des Lifecycles einzubinden

```
var vm = new Vue({  
  el: "#app",  
  data: { count: 0 }  
  created: function() {  
    this.count++;  
  }  
});
```



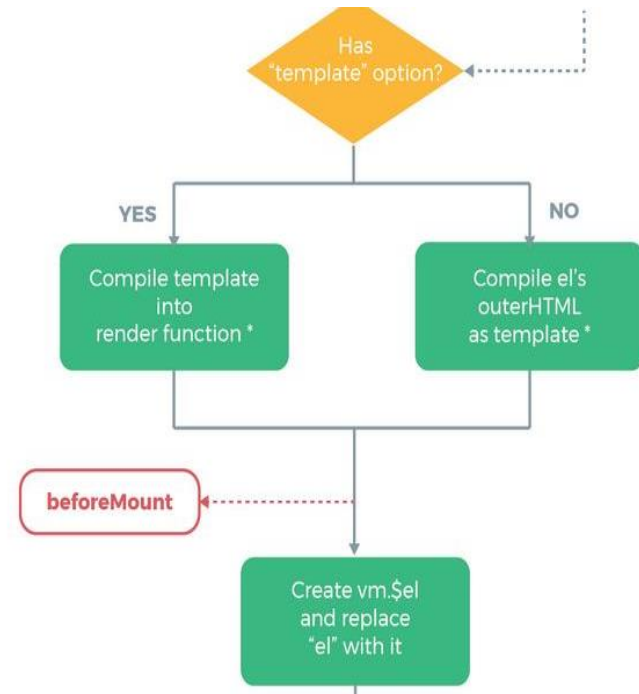
Quelle: <https://vuejs.org>

Vue.js Instances - Lifecycles

Eine Vue Instance durchläuft mehrere Initialisierungsschritte bevor sie erstellt wird

- Vue Instances erlauben über Lifecycle Hooks den Zugriff auf diese Schritte
- Hooks können dem options Objekt hinzugefügt werden, um Benutzer-spezifischen Code an diesem Schritt des Lifecycles einzubinden

```
var vm = new Vue({  
  el: "#app",  
  data: { count: 0 }  
  beforeMount: function() {  
    this.count++;  
  }  
});
```



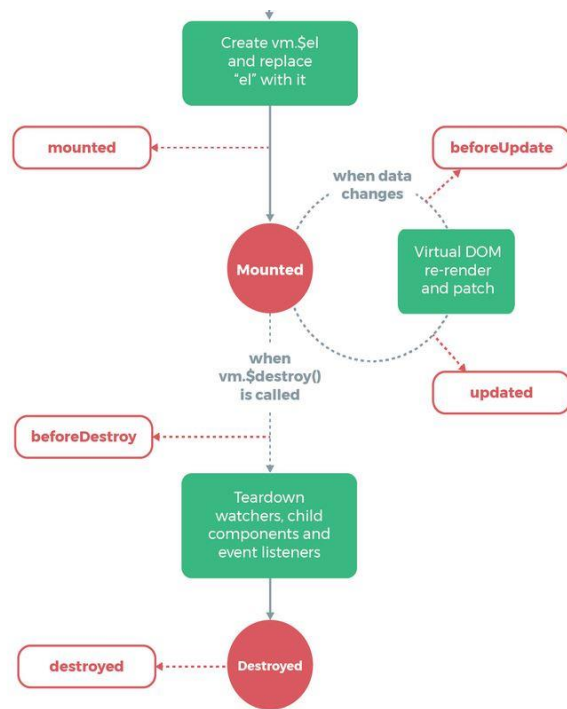
Quelle: <https://vuejs.org>

Vue.js Instances - Lifecycles

Eine Vue Instance durchläuft mehrere Initialisierungsschritte bevor sie erstellt wird

- Vue Instances erlauben über Lifecycle Hooks den Zugriff auf diese Schritte
- Hooks können dem options Objekt hinzugefügt werden, um Benutzer-spezifischen Code an diesem Schritt des Lifecycles einzubinden

```
var vm = new Vue({  
  el: "#app",  
  data:{ count: 0}  
  mounted: function(){  
    this.count++;  
  }  
});
```



Quelle: <https://vuejs.org>

Vue.js Templates

Vue.js verwendet eine HTML-basierte Vorlagensyntax (**Templates**), mit der das gerenderte HTML Dokument deklarativ an die Daten der zugrunde liegenden Vue Instance gebunden werden kann

- Alle Vue Templates sind gültige HTML Dokumente die von spezifikationskonformen Browsern und HTML Parsern übersetzt werden können
- Templates werden von Vue in Virtual DOM-Renderfunktionen kompiliert
- In Kombination mit dem Reaktivitätssystem kann Vue die minimale Anzahl von Komponenten ermitteln, die neu gerendert werden müssen, und so die minimale Anzahl von DOM-Manipulationen anwenden, wenn eine Änderung registriert wird

Vue.js Templates - Interpolation

Vue Interpolation

Die einfachste Form einer Bindung zwischen dem HTML Dokument und der Vue Instance ist die Text Interpolation über die **Moustache Schreibweise** **{{doppelte geschweifte Klammern}}**

`<div id="app">` → Vue-Template Deklaration innerhalb eines DOM-Element

`{{message}}` → Bindet den Inhalt des div-Elements an ein message Attribut der Daten der Vue-App

`</div>`

Über die id oder Klasse können wir das HTML Element mit einer Vue-App verbinden

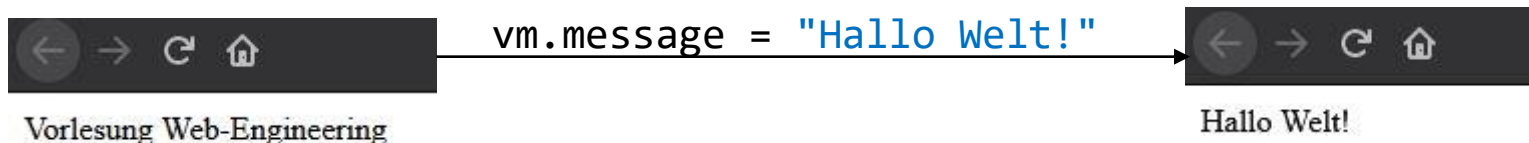
```
var vm = new Vue({  
  el: "#app",  
  data:{  
    message: "Vorlesung Web-Engineering"  
  }  
});
```



Vue.js Templates - Interpolation

Deklaratives Rendering

- Durch die erstellte Verbindung zwischen dem HTML Element und der Vue-App, ist dieses Element nun auch reaktiv
- Das heißt, wir können die Eigenschaften des HTML Elements während der Laufzeit manipulieren, ohne direkt auf die DOM zuzugreifen
- Wenn wir den angezeigten Text ändern wollen, können wir auf das message Attribute der Vue-App zugreifen und dieses verändern
- Die Änderung des message Attributes wird direkt auf der Webseite neu gerendert



Vue.js Templates

JavaScript Expressions

- Neben normalen Variablen können mithilfe der Text Interpolation auch reguläre JavaScript Ausdrücken an ein HTML Element gebunden werden

```
{{count + 1}}  
{{message.split("").reverse().join("")}}
```

- Eine Restriktion hierbei ist, das in jeder dieser Bindungen **nicht mehr als ein** Ausdruck verwendet werden kann

```
{{var count = 0}}  
  
{{  
  if (print)  
    {return message}  
}}}
```

Diese Ausdrücke sind **unglütig**

Vue.js Templates

Raw HTML

Bei der Text Interpolation interpretiert die Moustache Schreibweise seine enthaltenen Daten nur als `plain text` und nicht als HTML d.h.:

```
<div id="app">  
  {{rawHTML}}  
</div>
```



The diagram illustrates the rendering of raw HTML. On the left, a code snippet shows a Vue.js template with a `<div id="app">` containing `{{rawHTML}}`. A large curly bracket connects this code to a browser window on the right. The browser window shows the rendered output: a paragraph with the text "Roter Text" in red, corresponding to the raw HTML `<p style="color: red">Roter Text</p>` shown below the browser window.

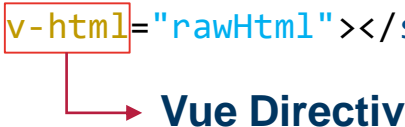
```
var vm = new Vue({  
  el: "#app",  
  data:{  
    rawHtml: "<p style= 'color: red'>Roter Text</p>"  
  }  
});
```

Vue.js Templates

Raw HTML

Um HTML mit Vue einzubinden werden statt der Text Interpolation sogenannte Direktives genutzt, hierbei wird der Inhalt des span Elements durch den des Attributes rawHTML ersetzt

```
<div id="app">  
  <span v-html="rawHtml"></span>  
</div>
```

 **Vue Directiv**



```
var vm = new Vue({  
  el: "#app",  
  data:{  
    rawHtml: "<p style= 'color: red'>Roter Text</p>"  
  }  
});
```

Vue.js Templates - Directives

Vue Directives ähneln im Wesentlichen HTML-Attributen, die Vue Templates hinzugefügt werden. Sie beginnen alle mit dem Prefix v-, um anzuzeigen, dass dies ein spezielles Vue Attribut ist.

- Die Aufgabe eines Directives besteht darin, reaktiv auf Änderungen von Werten innerhalb der Vue Instance zu reagieren, und diese nachträglich im HTML Element anzupassen
- Hierzu werden Directives Argumente übergeben, die das Verhalten des Directive bestimmen
- Die Werte eines Directives dürfen nur aus **einem** JavaScript Ausdruck bestehen mit Ausnahme von v-for
- Zwei der am häufigsten verwendeten Directives sind v-bind und v-on

Vue.js Templates - Directives

Die Directivs `v-bind` und `v-on`

- `v-bind` bindet ein oder mehrere Attribute an ein HTML Element
- Die Attribute werden als Argumente an das Directive übergeben

```

```

Bindet ein Attribut "someImg" als Pfad an das DOM-Element, der Pfad ist in diesem Beispiel das Argument

- `v-on` bindet einen EventHandler an ein HTML Element
- Als Argument erwartet `v-on` ein valides Event

```
<button v-on:click="doSomething">Ein Knopf</button>
```

Übergebenes Event bei dem das gegebene Attribute ausgeführt wird. Beim gegebenen Argument kann es sich auch um eine Funktion handeln

Vue.js Templates - Directives

Es ist ebenfalls möglich Dynamische Argumente (ab Vue Version 2.6.0+) an Directive zu übergeben

- Hierzu wird das Argument in eckige Klammern geschrieben

```
<a v-bind:[argument]="someUrl"></a>
```

- In diesem Beispiel wird das Argument argument innerhalb der Vue Instance nachgeschlagen und eingesetzt, wenn dieses Argument als Attribute in der Vue Instance vorhanden ist
- Analog können so auch dynamische Events mithilfe von v-on implementiert werden

```
<button v-on:[event]="doSomething">Ein Knopf</button>
```

Vue.js Templates - Directives

Einschränkungen bei Dynamischen Argumenten

- Dynamische Argumente müssen als String übergeben werden
 - Einzige Ausnahme ist hierbei der Wert **null**, welcher dafür genutzt werden kann eine Bindung zu löschen
 - Jeder andere Wert der nicht vom Typ String ist, gibt einen Fehler zurück

<!-- Dieser Code wird eine Warnung auswerfen -->

```
<a v-bind:[h + "ref"] = "someUrl"></a>
```


Vue.js Templates - Directives

Einschränkungen bei Dynamischen Argumenten

- Weiterhin gibt es zwei Syntax Einschränkungen bei der Verwendung von Dynamischen Argumenten
 - Bestimmte Zeichen, z.B. Leerzeichen und Anführungsstriche, sind ungültig innerhalb von HTML Attribute Namen
 - Großbuchstaben innerhalb der Namen sollten ebenfalls vermieden werden, da manche Browser diese in Kleinbuchstaben überführen

```
<!-- someAttr wird vom Compiler zu someattr übersetzt, sollte  
also kein Attribute mit dem Namen someattr innerhalb der Vue  
Instance existieren wird der Code nicht funktionieren-->
```

```
<a v-bind:[someAttr] = "someUrl"></a>
```

Vue.js Templates - Directives

Für die beiden häufigsten Directives, `v-bind` und `v-on`, gibt es HTML-konforme Kurzschreibweisen

- `v-bind` kann mit einem `:` abgekürzt werden

```
<!-- Volle Syntax-->  
<a v-bind:[path] = "someUrl"></a>  
<!-- Verkürzster Syntax-->  
<a :[path] = "someUrl"></a>
```

- `v-on` wird mit einem `@` abgekürzt

```
<!-- Volle Syntax-->  
<a v-on:[event] = "doSomething"></a>  
<!-- Verkürzter Syntax-->  
<a @[event] = "doSomething"></a>
```

- Die Zeichen `:` und `@` sind beide valide HTML Zeichen die von Vue kompatiblen Browsern interpretiert werden können

Vue.js Computed Properties

Berechnungen innerhalb von Vue Templates über die Moustache Schreibweise sind zwar sehr Hilfreich, aber sind nur für simple Operationen gedacht

```
{{message.split("").reverse().join("")}}
```

- Werden diese Operationen zu komplex, wird der Code unübersichtlich und schwerer zu warten
- Für Operationen wie im oberen Beispiel braucht es manchmal einen zweiten Blick um zu erkennen, das hier der Inhalt von message Rückwärts angezeigt wird
- Um dies zu vermeiden können Computed Properties genutzt werden, dies bietet sich besonders an, wenn eine Operation mehrmals auf dem gleichen oder unterschiedlichen Elementen ausgeführt werden soll

Vue.js Computed Properties

Computed Properties werden über das options Attribute computed in eine Vue Instance eingebunden

```
<div id="app">  
  <p>Ohne Computed Properties: {{message}}</p>  
  <p>Mit Computed Properties: {{reversedMessage}}</p>  
</div>
```

```
var vm = new Vue({  
  el:"#app",  
  data:{  
    message: "Vorlesung Web-Engineering"  
  },  
  computed:{  
    reversedMessage: function(){  
      return this.message.split("").reverse().join("");  
    }  
  }  
});
```

Ohne Computed Properties: Vorlesung Web-Engineering

Mit Computed Properties: gnireenignE-beW gnuselroV

Vue.js Computed Properties

Computed Properties vs. Methods

- Auf den ersten Blick unterscheiden sich Computed Properties kaum von den zuvor vorgestellten methods, da beide das gleich Ergebnis zurückgeben würden

```
<div id="app">
  <p>Mit Methods: {{reversedMessage()}}</p>
</div>
```

...

```
methods:{
  reversedMessage: function(){
    return this.message.split("").reverse().join("")
  }
}
```

Mit Methods: gnireenignE-beW gnuselroV

Vue.js Computed Properties

Computed Properties vs. Methods

- Generell wird bei jedem Funktionsaufruf die Funktion ausgeführt und ein Ergebnis zurückgegeben, während dies bei methods weiterhin zutrifft, verwenden Computed Properties sogenannte caches, welche die Ergebnisse der letzten Operation speichern
- Computed Properties reagieren auf Änderung ihrer referenzierten Attribute, im vorherigen Beispiel message
- Sollte sich message seit dem letzten Funktionsaufruf nicht geändert haben, wird das gespeicherte (cached) Ergebnis verwendet, ohne dass die Funktion erneut ausgeführt wird
- Dies ist besonders nützlich bei größeren und komplexeren Funktionen um Rechenzeit zu sparen

Vue.js Klassen- und Stilbindungen

Mithilfe des **v-bind Directives** können wir die Klassen und Stile von **DOM-Elementen** dynamisch anpassen

- **v-bind** erlaubt es uns nicht nur Strings als Parameter zu übergeben, sondern auch Objekte und Arrays

```
<div v-bind:class="positiv: isPositiv"></div>
```

- Hierbei gibt **isPositiv** über seinen Wahrheitswert die Existenz der Klasse **positiv** an
- **isPositiv** muss dementsprechend in einer **data** Eigenschaft der Vue Instance sein

Vue.js Klassen- und Stilbindungen

Klassenbindung

- Es können ebenfalls mehrere Klassen innerhalb eines Elements eingefügt werden

```
<div v-bind:class="positiv: isPositiv, negativ: isNegativ"></div>
```

- Je nach Wahrheitswert der data Eigenschaften ergibt sich die Klasse des div Elementes
- Sollten beide Wahrheitswerte im oberen Beispiel true sein, wird folgende Klasse erstellt

```
<div class="positiv negativ"></div>
```


Vue.js Klassen- und Stilbindungen

Klassenbindung

- Anstatt inline Argumente zu nutzen, kann auch ein Objekt über v-bind übergeben werden

```
<div v-bind:class="classObject"></div>
```

```
data:{  
  classObject: {  
    positiv: true,  
    negativ: false  
  }  
}
```

- Die erzeugte Klasse dieses Beispiels ist:

```
<div class="positiv"></div>
```

Vue.js Klassen- und Stilbindungen

Stilbindung

Analog zu dem Binden von Klassen, können auch CSS-Stile an HTML Elemente gebunden werden

- Entweder über inline Argumente:

```
<div v-bind:style="{color: myColor, fontSize: fontSize + 'px'}"></div>
```

- Oder über ein Objekt:

```
<div v-bind:style="styleObject"></div>
```

```
data:{
  styleObject: {
    color: "red",
    fontSize: "12px"
  }
}
```

Vue.js Conditional Rendering

Um ein HTML Element nur dann zu erstellen wenn es benötigt wird, kann das Directive `v-if` verwendet werden

- Das Directive `v-if` erstellt ein HTML Element nur dann, wenn sein Wahrheitswert `true` ist

```
<h1 v-if="easy">Web-Engineering ist leicht</h1>
```

- Mit `v-else` kann eine else-Klausel erstellt werden

```
<h1 v-if="easy">Web-Engineering ist leicht</h1>  
<h1 v-else>Web-Engineering ist schwer</h1>
```

- Seit Vue Version 2.1.0 existiert auch das Directive `v-else-if`

```
<h1 v-if="easy">Web-Engineering ist leicht</h1>  
<h1 v-else-if="medium">Web-Engineering ist anspruchsvoll</h1>  
<h1 v-else>Web-Engineering ist schwer</h1>
```

Vue.js Conditional Rendering

Da `v-if` und seine Partner `v-else-if` und `v-else` Directives sind, können Sie nur einem HTML Element angehängen werden

Wenn mehrere Elemente von einer Bedingung abhängig sein sollen, kann das Element `<template>` genutzt werden, dieses dient als unsichtbarer Block

```
<template v-if="show">
  <h1>Web-Engineering</h1>
  <p>Thema</p>
  <p>Vue.js</p>
</template>
```

Vue.js List Rendering

Um eine Liste von Elementen mit Vue zu erstellen, wird das Directive **v-for** genutzt

Hierbei wird die aus JavaScript bekannte Form von **item in items** genutzt

```
<ul id="vuelist">
  <li v-for="item in items">
    {{item.message}}
  </li>
</ul>
var vm = new Vue({
  el: "#vuelist",
  data: {
    items: [
      {message: "Vorlesung 1"},
      {message: "Vorlesung 2"}
    ]
  }
});
```

- Vorlesung 1
- Vorlesung 2

Vue.js List Rendering

Mit v-for ist es ebenfalls möglich über ein Objekt zu iterieren

```
<ul id="vuelist">
  <li v-for="value in object">
    {{value}}
  </li>
</ul>var vm = new Vue({
  el:"#vuelist",
  data:{
    object: {
      name: "Alice",
      message: "Hallo Welt"
    }
  }
});
```

- Alice
- Hallo Welt

Vue.js - Eventhandling

Um auf Ereignisse (Events) innerhalb der Webseite zu reagieren, nutzt Vue ein Eventhandling über v-on

```
<div id="app">  
  <button v-on:click="counter += 1">Drück mich!</button>  
  <p>Der Knopf wurde {{ counter }} oft gedrückt.</p>  
</div>
```

```
var vm = new Vue({  
  el: '#app',  
  data: {  
    counter: 0  
  }  
});
```

Bei einem Mausklick wird eine Operation ausgeführt, welche die variable counter erhöht

Vue.js - Eventhandling

Bei größeren Operationen ist es besser, direkt auf eine Methode oder ein Computed Property zu verweisen, welche beim Eintreten des Events ausgeführt wird

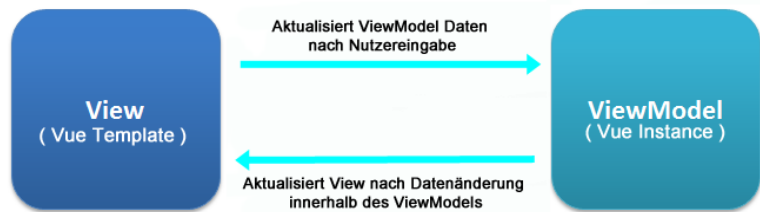
```
<div id="app">
  <button v-on:click="rndNumber">Zufallszahl</button>
  <p>Die Zufallszahl ist {{ number }}</p>
</div>
var vm = new Vue({
  el: '#app',
  data: {
    number: 0
  },
  methods: {
    rndNumber: function(){
      this.number = Math.floor(Math.random() * 100)
    }
  }
});
```


Vue.js - Benutzereingaben

Für Benutzereingaben wird eine zwei-Wege Bindung zwischen dem Template und der Vue Instance benötigt

- Um diese Verbindung zu erstellen wird das Directive `v-model` verwendet
- `v-model` ermöglicht die stetige Aktualisierung der Daten bei Benutzereingaben

```
<div id="app">
  <input v-model="message" placeholder="Schreib etwas">
  <p> Deine Eingabe: {{ message }}</p>
</div>
var vm = new Vue({
  el:"#app",
  data:{
    message:""
  }
});
```



Vue.js - Benutzereingaben

v-model Eigenschaften

- v-model ignoriert bei der Initialisierung vorhandene Werte innerhalb des Eingabefeldes, da es die Daten in der Vue Instance als „source of truth“ betrachtet
 - Wird ein initialer Wert im Eingabefeld benötigt, muss dieser in der Vue Instance initialisiert werden
- Für unterschiedliche Eingabefeldtypen nutzt v-model unterschiedliche Eigenschaften und Events

Eingabefeld Typ	Eigenschaft	Event
Text, Textarea	value	input
Checkboxes, Radiobuttons	checked	change
Select Fields	value	change