

RUHR-UNIVERSITÄT BOCHUM

WEB-ENGINEERING

Sommersemester 2023

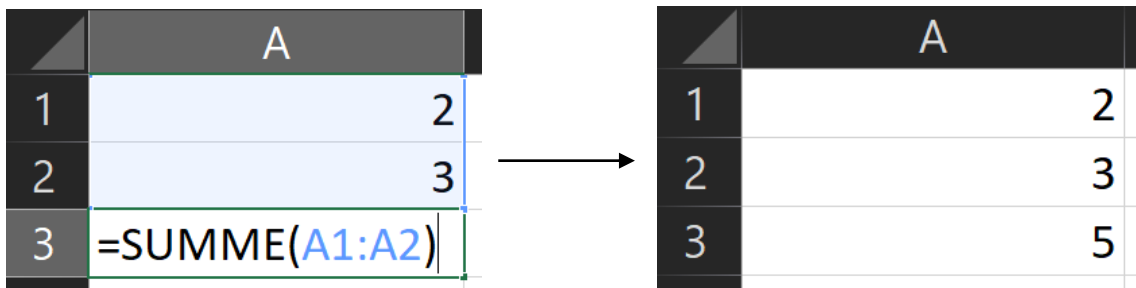


Informatik
im Bauwesen

Vue.js Reactivity

Was ist Reactivity?

- Reaktivität ist ein Programmierparadigma, das uns erlaubt, **deklarativ auf Änderungen reagieren zu können**
- Zur besseren Veranschaulichung können wir uns ein einfaches Excel-Tabellen Beispiel anschauen



| | A |
|---|---------------|
| 1 | 2 |
| 2 | 3 |
| 3 | =SUMME(A1:A2) |

| | A |
|---|---|
| 1 | 2 |
| 2 | 3 |
| 3 | 5 |

Vue.js Reactivity

Was ist Reactivity?

- Zeile 3 bildet in diesem Beispiel die Summe aus den Werten der ersten und zweiten Zeile
- Wenn wir nun den Wert aus Zeile 1 ändern wird Zeile 3 ohne unser Zutun automatisch angepasst, Zeile 3 ist also reaktiv

| | A |
|---|---|
| 1 | 2 |
| 2 | 3 |
| 3 | 5 |

| | A |
|---|---|
| 1 | 4 |
| 2 | 3 |
| 3 | 7 |

Vue.js Reactivity

Was ist Reactivity?

- Das zuvor gezeigte Beispiel zur Reaktivität ist ohne weiteres so in JavaScript nicht umsetzbar
- Betrachten wir hierfür folgendes Code-Beispiel
- Wenn wir den ersten Wert verändern, wird die Summe nicht aktualisiert

```
let zahl1 = 2;  
let zahl2 = 3;  
let sum = zahl1 + zahl2;  
  
console.log(sum); // 5  
zahl1 = 4;  
console.log(sum); // bleibt 5
```

Vue.js Reactivity

Was ist Reactivity?

- Um die im Excel vorgestellte Reaktivität umzusetzen, müssen in JavaScript folgende Punkte umgesetzt werden:
 1. Überwachen, wenn ein Wert vom Programm gelesen wird. z.B. `zahl1 + zahl2` liest sowohl `zahl1` und `zahl2`
 2. Feststellen, wann sich ein Wert ändert. z.B. Wenn `zahl1 = 3` geändert wird.
 3. Erneutes ausführen aller Operationen die diesen Wert ausgelesen haben. z.B. `sum = zahl1 + zahl2`

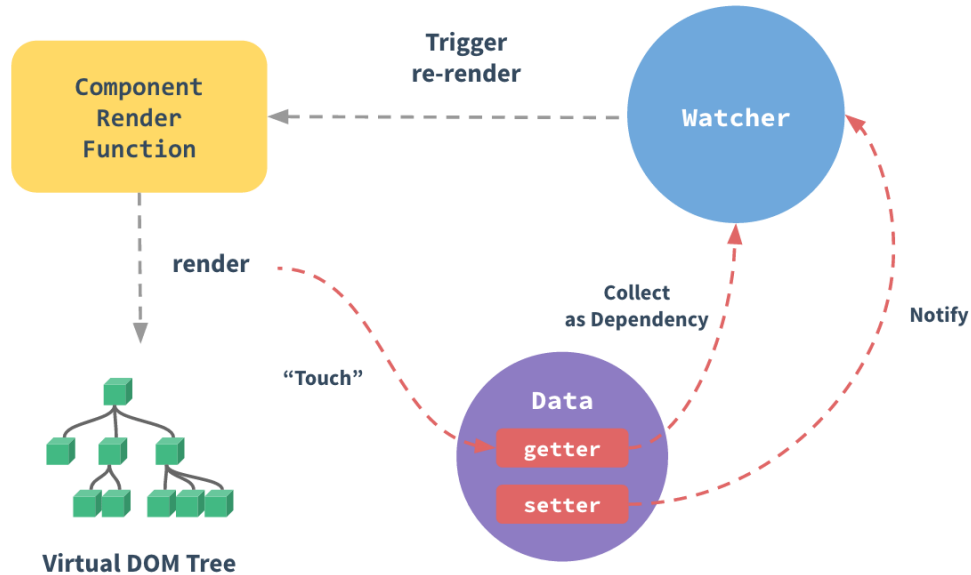
Vue.js Reactivity

Wie funktioniert Reactivity in Vue?

- Eine Möglichkeit Reaktivität in JavaScript zu nutzen haben wir bereits in der Moustache Schreibweise `{{}}` von Vue kennengelernt
- In Vue werden die zuvor genannten 3 Punkte durch Getter und Setter Methoden umgesetzt
- Jede Eigenschaft innerhalb des data Objektes der Vue Instanz wird zu `gettern` und `settern` konvertiert
- Hierfür wird eine Funktion `Object.defineProperty` genutzt, welche es erst seit ES5 gibt und somit nicht in IE8 oder älter genutzt werden kann
- Zusätzlich wird ein Beobachter (Watcher) erstellt, welcher auf Änderungen im data Objekt reagiert

Vue.js Reactivity

Wie funktioniert Reactivity in Vue?



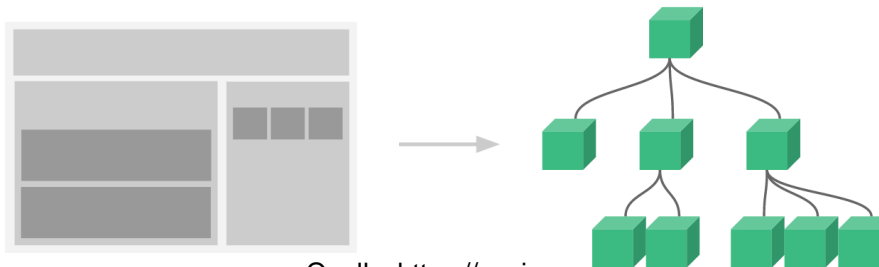
Web-Engineering

Vue.js Components

Vue.js Components

Basics

- Bisher haben wir für die Erstellung von Webseiten mit Vue nur eine Vue Instance erstellt
- Oft sollten Webseiten aber in kleinere Teile (Kopfzeile, Hauptteil, Fußzeile) aufgeteilt werden, hierfür können Vue Components genutzt werden
- Die bisher genutzte Vue Instance ist in diesem Fall als oberste (root) Instanz bekannt



Quelle: <https://vuejs.org>

Vue.js Components

Basics

- Vue Components sind wiederverwendbare Vue Instances
- Beim Erstellen einer Vue Component wird als erstes Argument immer der **Name der Component** festgelegt
- Es sollte darauf geachtet werden, das beim Vergeben der Namen auf die Richtlinien für custom html tags (**Kleingeschrieben und mindestens einen Bindestrich**) geachtet wird

```
Vue.component("component-name",{  
  //options  
});
```

Vue.js Components

Basics

- Da Vue Components auch Vue Instances sind, akzeptieren sie die gleichen Optionen, z.B. data, computed oder methods
- Eine Ausnahme gilt für Optionen die nur in der root Vue Instanz genutzt werden können, z.B. el

```
Vue.component("component-name",{  
  //options  
});
```

Vue.js Components

Basics

- Ein weiterer Unterschied zwischen der root Instanz und den Vue Components ist das **data Attribut**
- In der root Instanz kann hier ein einfaches Objekt mit Daten übergeben werden
- In Components muss das data Attribut allerdings eine Funktion enthalten

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

```
Vue.component("component-name",{  
  data: function(){  
    return {count: 0}  
  }  
});
```

Vue.js Components

Basics

- Durch die Nutzung von Funktionen innerhalb des data Attributes, können **mehrere Instanzen einer Vue Component** ihre Werte unabhängig voneinander verändern
- Eine Vue **Component** wird **über einen custom html tag** mit dem deklarierten Namen der Component dem HTML Dokument hinzugefügt

```
Vue.component("component-  
name",{  
  data: function(){  
    return {count: 0}  
  }  
});
```

```
<div id="app">  
  <component-name></component-name>  
</div>
```

Achtung:

Wenn data keine Funktion enthält,
wirft Vue einen Fehler aus

Vue.js Components

Basics – Beispiel

```
Vue.component("button-counter", {  
  data: function(){  
    return {counter: 0}  
  },  
  methods:{  
    count: function(){this.counter++;}  
  },  
  template: '<button v-on:click="count">Der  
Knopf wurde {{counter}}  
gedrückt.</button>'  
});
```

Vue Component (JS)

```
var vm = new Vue({  
  el: "#app"  
});
```

Vue Instance (JS)

```
<div id="app">  
  <button-counter></button-counter>  
  <button-counter></button-counter>  
  <button-counter></button-counter>  
</div>
```

Vue Template (HTML)

Vue.js Components

Exkurs HTML Custom Tags

- Mit Webcomponents lassen sich eigene HTML-Elemente erstellen.
- Der Code wird kürzer, lesbarer und lässt sich besser Strukturieren. Zudem ist eine Wiederverwendbarkeit von Abschnitten gegeben.
- Die Elemente werden dabei sowohl im HTML Code als auch im Javascript Code hinterlegt. Eine Definition in HTML sieht beispielsweise so aus:
`<my-element> ... </my-element>` - damit dieses jedoch funktioniert muss die Funktion in einer Klasse definiert werden

Vue.js Components

Exkurs HTML Custom Tags

- Die Klasse muss einen Konstruktor und bei Bedarf einen Getter haben. Nützliche Methoden zu der Erstellung eigener Tags finden sich in der Dokumentation
- Dazu gibt es einige Konventionen und Richtlinien die man beachten sollte
- Namen von benutzerdefinierten Elementen müssen immer einen Bindestrich enthalten - damit bleiben sie auch mit neuen HTML-Standard und neuen Elementen kompatibel

Vue.js Components

Exkurs HTML Custom Tags

- Ein Tag kann **nur einmal registriert** werden, wenn ein Tag mehrfach registriert werden soll wird ein Fehler ausgegeben
- Benutzerdefinierte Elemente müssen immer **mit einem schließenden Tag geschlossen** werden.
- Beispiel:

```
<my-element> ... </my-element>  
class myElement extends HTMLElement {  
  ...  
}  
customElements.define('my-element', myElement);
```

Vue.js Components

Registrierung von **globalen** Vue Components

- In dem zuvor gezeigten Beispiel wurde die Vue Component global registriert
- Durch den Aufruf von `Vue.component()` **wird eine Component global registriert**
- Eine globale Vue Component kann in jedem Template einer root Vue Instanz und in anderen Vue Components genutzt werden

```
Vue.component("component-a",...);  
Vue.component("component-b",...);  
Vue.component("component-c",...);  
var vm = new Vue(...);
```

```
<div id="app">  
  <component-a></component-a>  
  <component-b></component-b>  
  <component-c></component-c>  
</div>
```

Vue.js Components

Registrierung von **lokalen** Vue Components

- Alternativ zur globalen Registrierung von Vue Components, gibt es auch eine lokale Registrierung
- Lokale Vue Components können nur innerhalb ihrer deklarierten Vue Instanz oder Vue Component genutzt werden
- Hierbei wird die Vue Component als JS-Objekt erstellt und über das `options` Attribut `components` bei der gewünschten Instanz registriert
- Der Key repräsentiert hier den Namen des custom `html` tags, während das `Value` das erstellte JS Objekt enthält

Vue.js Components

Registrierung von **lokalen** Vue Components

```
var ComponentA = { /*options*/ };  
var ComponentB = { /*options*/ };  
var ComponentC = { /*options*/ };
```

```
var vm = new Vue({  
  components: {  
    "component-a" = ComponentA,  
    "component-b" = ComponentB,  
    "component-c" = ComponentC  
  }  
});
```

KEY VALUE

Vue.js Components

Registrierung von lokalen Vue Components

- Lokal registrierte Components können nicht von anderen Components genutzt werden, solange Sie nicht ebenfalls mit der `components` Option innerhalb der Component registriert werden

```
var ComponentA = { /*options*/ };  
var ComponentB = {  
  components: {  
    "component-a" = ComponentA  
  };  
};
```

Vue.js Components

Registrierung von lokalen Vue Components

- Lokal registrierte Components sollten global registrierten Components vorgezogen werden da:
 - Besonders bei großen Projekten zu viele Abhängigkeiten zu globalen Components entstehen, was den Code schwerer wartbar macht
 - Zudem müssen bei Unit-Test immer sämtliche globalen Components mit initialisiert werden, auch wenn sie nicht benötigt werden

Vue.js Components

Props

- Um Components Werte zu übergeben werden sogenannte Props verwendet
- Props werden über das options Attribute props einer Componente hinzugefügt
- Die template Option liefert ein HTML Dokument, welches beim Einbinden der Component dem ursprünglichen HTML Dokument zugefügt wird

```
Vue.component("component-a",{  
  props:["titel"],  
  template: "<h1>{{titel}}</h1>"  
});
```

Vue.js Components

Props

- Die einfachste Methode einem Prop Daten zuzuweisen, ist über eine statische Zuweisung
- Hierfür wird der Wert des Props über das Template direkt an die Component übermittelt
- Diese Methode eignet sich nur für statische Elemente einer Webseite

```
Vue.component("component-a",{  
  props:["title"],  
  template: "<h1>{{title}}</h1>"  
});
```

```
<component-a title="Hello"></component-a>  
<component-a title="World"></component-a>
```


Vue.js Components

Props

- Das props Attribute kann ebenfalls mehr als ein Element enthalten, hierbei werden alle weiteren Elemente in ein Array abgelegt
- Sollen alle Elemente innerhalb der `template` Option angezeigt werden, muss darauf geachtet werden, dass die Elemente einem oberen Element zugeordnet werden z.B. `<div>`
- Dies ist notwendig, da jede Component nur ein Element enthalten darf

Vue.js Components

Props

```
Vue.component("component-a",{
  props:["titel", "content", "date"],
  template:"<h1>{{titel}}</h1>
    <p>{{date}}</p>
    <p>{{content}}</p>"
});
```



Gibt einen Fehler zurück

```
Vue.component("component-a",{
  props:["titel", "content", "date"],
  template:'<div class="component-a">
    <h1>{{titel}}</h1>
    <p>{{date}}</p>
    <p>{{content}}</p>
  </div>'
});
```

Vue.js Components

Props

- Statische Zuweisungen können auch **andere Datentypen** übergeben wie z.B. Booleans und Arrays
- Hierzu wird das Directive `v-bind` benötigt, um Vue zu zeigen, das es sich um einen JavaScript Ausdruck handelt und nicht um einen String
- Es ist auch möglich, mehrere Props nicht als ein Array von Strings zu übergeben, sondern **als Objekt** mit dem jeweiligen Attributnamen und Typen

Vue.js Components

Props

```
<component-a v-bind:bool="false"></component-a>  
<component-a v-bind:narray="[1,2,3]"></component-a>
```

```
Vue.component("component-a",{  
  props:{  
    bool: Boolean,  
    narray: Array}  
});
```

Vue.js Components

Dynamische Props

- Sollte der Inhalt einer Component nicht bei der Initialisierung bekannt sein oder sich dieser später auch ändern können, müssen die Daten dynamisch an die Component übergeben werden
- Hierfür wird das Directive `v-bind` genutzt, um die Props der Component an das data Elemente der root Instanz zu binden

Vue.js Components

Dynamische Props

```
var vm = new Vue({  
  el: "#app"  
  data:{  
    propData:{  
      title: "Hello World",  
      date: "06.07.2020",  
      content: "Lorem Ipsum"}  
    }  
});
```

```
<div id="app">  
  <component-a v-bind:title="propData.title"  
    v-bind:date="propData.date"  
    v-bind:content="propData.content">  
  </component-a>  
</div>
```

Vue.js Components

Dynamische übergebene Objekte an Props

- Die zuvor gezeigte Schreibweise funktioniert gut bei wenigen Elementen innerhalb der Component, bei mehreren Elementen wird diese Schreibweise allerdings **sehr unübersichtlich**
- Stattdessen kann **auch ein Objekt als Prop** übergeben werden, welches von der Component ausgelesen werden kann
- Dies ermöglicht auch, **dem** übergebenen Objekt **innerhalb der root Instanz eine neue Eigenschaft hinzuzufügen** und diese in der Component anschließend nutzen zu können

Vue.js Components

Prop Casing

- Browser interpretieren Großbuchstaben immer als Kleinbuchstaben, da HTML Attributnamen nicht Groß- und Kleinschreibung beachten (case-insensitive)
- Aus diesem Grund müssen **camelCase** Propnamen innerhalb von DOM-Templates, bzw. innerhalb des HTML Dokumentes, mit ihrem *kebab-case* äquivalent geschrieben werden

HTML:

```
<component-a v-bind:prop-  
data="propData"></component-a>
```

JS:

```
Vue.component("component-a", {  
  props: ["propData"]  
})
```


Vue.js Components

Beispiel

```
Vue.component("component-a", {  
  props: ["prop-data"],  
  template: '<div class="component-a">  
    <h1>{{propData.title}}</h1>  
    <p>{{propData.date}}</p>  
    <p>{{propData.content}}</p>  
  </div>'  
});
```

Vue Component (JS)

```
var vm = new Vue({  
  el: "#app",  
  data: {  
    propData: {  
      title: "Hello World",  
      date: "06.07.2020",  
      content: "Lorem Ipsum"}  
    }  
  }  
});
```

Vue Instance (JS)

```
<div id="app">  
  <component-a  
    v-bind:prop-data="propData">  
  </component-a>  
  <component-a  
    v-bind:prop-data="propData">  
  </component-a>  
</div>
```

Vue Template (HTML)

Vue.js Components

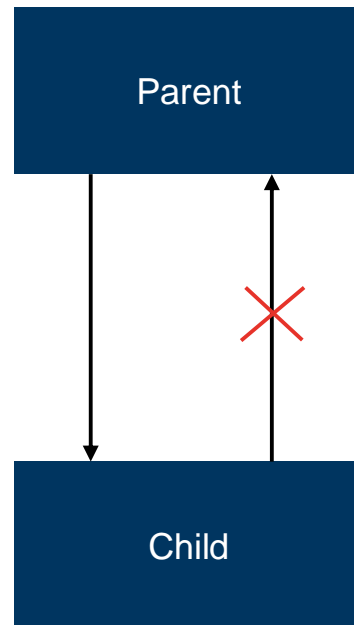
One-way Data Flow

- Alle Props besitzen einen **One-way Data Flow** zwischen dem Child Component und dem Parent Component
- Daten werden vom Parent Component an das Child Component weitergereicht, aber nicht andersherum
- Zusätzlich wird das Child Component bei einer relevanten Änderung innerhalb des Parent Component automatisch aktualisiert
- Sollte ein Prop innerhalb einer Child Component geändert werden, wirft Vue eine Warnung aus

Vue.js Components

One-way Data Flow

Achtung: Objekte und Arrays funktionieren in JavaScript über *Call by Reference*. Änderungen an übergebenen Arrays oder Objekten in der Child Component haben eine Auswirkung auf das Parent Component



Vue.js Components

One-way Data Flow

- Das Verändern von Props innerhalb der Child Component sollte vermieden werden, allerdings gibt es Szenarien in denen es sich anbieten würde:
 1. Das Prop soll genutzt werden, um einen initialen Wert in das Child Component zu übergeben
 - In diesem Fall sollte eine lokale Variable erstellt werden, die den Wert des übergebenen Props speichert

```
props: ["initialProp"],  
  data():{  
    return{  
      prop: this.initialProp  
    }  
  }  
}
```

Vue.js Components

One-way Data Flow

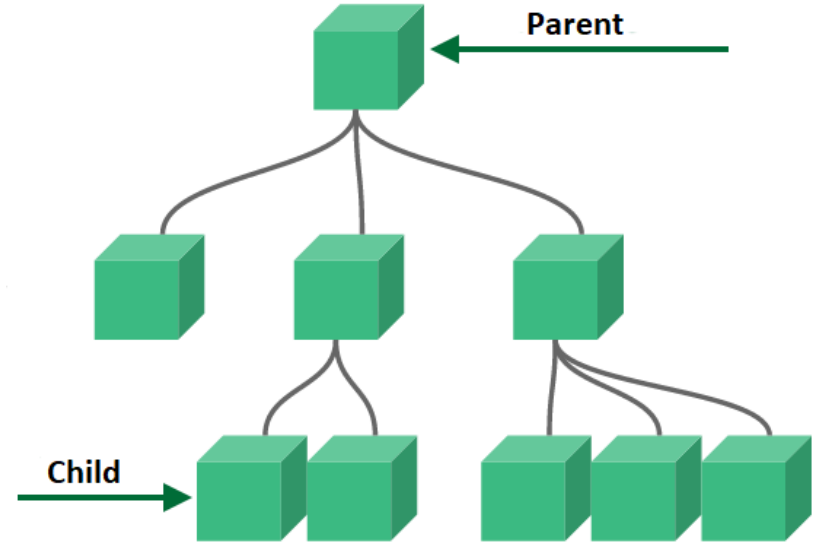
- Das Verändern von Props innerhalb der Child Component sollte vermieden werden, allerdings gibt es Szenarien in denen es sich anbieten würde:
- 2. Es sollen Rohdaten an die Child Component übergeben werden, die in dieser verarbeitet werden sollen
- Als Lösung kann hier ein computed property genutzt werden, welches den Wert des übergebenen Props nutzt

```
props: ["size"],
computed: {
  normalizedSize: function() {
    return this.size.trim().toLowerCase()
  }
}
```

Vue.js Components

Prop Provide & Inject

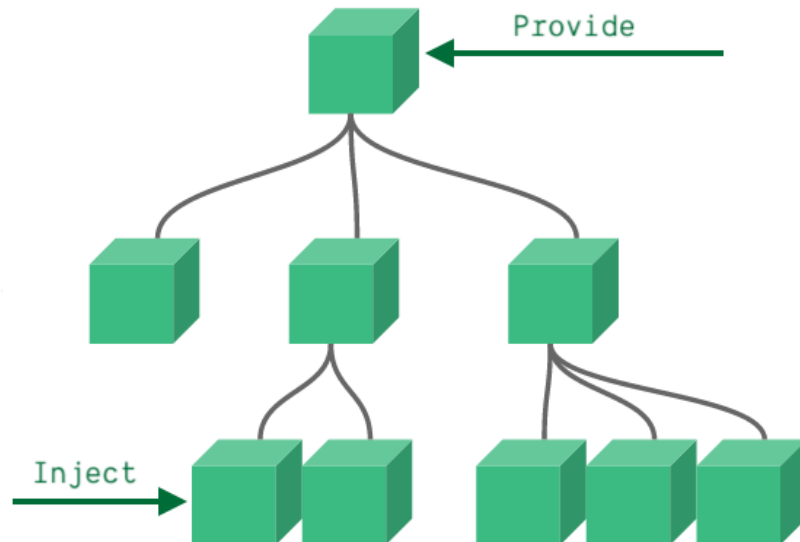
- Wie bereits gezeigt, werden Props verwendet, um Daten von Parent Components zu Child Components weiterzureichen
- Dies wird Problematisch, sobald wir uns mehrere Stufen von der Parent Component entfernen, da in diesem Fall, die Daten über mehrere Child Components nach unten gereicht werden müssten



Vue.js Components

Prop Provide & Inject

- Für dieses Problem stellt Vue die `provide` und `inject` Optionen zur Verfügung
- Eine Parent Component die die `provide` Option besitzt, kann so jede Child Component unter ihm ansprechen, egal wie tief diese in der Hierarchie verankert ist



Vue.js Components

Prop Provide & Inject

- Schauen wir uns folgendes Beispiel an:
- Wenn wir nun nur über Props Daten von `TodoList` nach `TodoListStatistics` schicken wollen, müssen wir diese über `TodoList` -> `TodoListFooter` -> `TodoListStatistics` weiterreichen
- Mithilfe von `provide` und `inject` können wir dies direkt tun

```
Root
└─ TodoList
    │   └─ TodoItem
    │   └─ TodoListFooter
    │       │   └─ ClearTodosButton
    │       │   └─ TodoListStatistics
```


Vue.js Components

Prop Provide & Inject

Parent Component:

```
Vue.component('todo-list', {  
  data() { return {  
    todos: ['Feed a cat',  
            'Buy tickets'] }},  
  provide: { user: 'John Doe' }  
})
```

Child Component:

```
Vue.component('todo-list-statistics', {  
  inject: ['user'],  
  created() {  
    console.log(`Injected  
property: ${this.user}`)  
  }  
})
```

Vue.js Components

Prop Provide & Inject

- Das vorherige Beispiel funktioniert nur mit neu erstellten Objekten, die nicht in der data Option der Component hinterlegt sind
- Wenn Eigenschaften der data Option weitergereicht werden sollen, muss provide ebenfalls zu einer Funktion konvertiert werden
- Provide & Inject **stellen standardmäßig keine Reaktivität zwischen Parent und Child Component her**

```
Vue.component('todo-list', {  
  data() { return {  
    todos: ['Feed a cat',  
            'Buy tickets'] }},  
  provide():{ return {  
    todoLength: this.todos.length}}  
})
```

Vue.js Components

Reagieren auf Events in Child Components

- Es wurde bereits gezeigt das über die methods Option eine Funktionalität zu einer Component hinzugefügt werden kann
- Mit dieser Methode werden aber nur Daten innerhalb der Componente manipuliert
- Wenn eine Nutzerinteraktion Einfluss auf mehrere Component Instanzen haben soll, muss die Component mit der überliegenden Instanz kommuniziert
- Hierfür wird die Vue Methode `$emit` genutzt

```
<button v-on:click="$emit('enlarge-text')">Vergrößere Text</button>
```

Vue.js Components

Reagieren auf Events in Child Components

- Über `$emit` wird das `Custom Event enlarge-text` ausgelöst
- Die überliegende Instanz kann dann auf das Event reagieren

```
<component-a v-on:enlarge-text="propFontSize += 0.1"></component-a>
```

- Im oberen Beispiel wird die Größe des Textes statisch um einen Faktor von 0.1 erhöht, dies kann ebenfalls dynamisch gestaltet werden

Vue.js Components

Reagieren auf Events in Child Components

- Der Methode `$emit` kann ein zweiter Parameter übergeben werden, der beim auslösen mitgesendet wird

```
<button v-on:click="$emit('enlarge-text', 0.1)">Vergrößere Text</button>
```

- Auf den zweiten Parameter kann innerhalb der übergeordneten Instanz mit der Methode `$event` zugegriffen werden

```
<component-a v-on:enlarge-text="propFontSize += $event"></component-a>
```

Vue.js Components

Custom Event Casing

- Browser interpretieren Großbuchstaben immer als Kleinbuchstaben, da HTML Attributnamen nicht Groß- und Kleinschreibung beachten (case-insensitive)
- Ähnlich wie bei Props, müssen *camelCase* Eventnamen innerhalb von DOM-Templates, bzw. innerhalb des HTML Dokumentes, mit ihrem *kebab-case* äquivalent geschrieben werden

HTML: `<component-a v-on:my-event="doSomething"></component-a>`

JS: `this.$emit("myEvent")`

Vue.js Components

Erstellen von Custom Events

- Custom Events können innerhalb einer Component über die `emits` Option definiert werden

```
Vue.component('custom-form', {  
  emits: ['myEvent', 'submit']  
})
```

- Standard Events (z.B. `click`) die innerhalb der `emits` Option erneut deklariert werden, werden durch das in `emits` Option deklarierte Event überschrieben
- Wenn eine Component mehrere Custom Events besitzt, ist die `emits` Option zu empfehlen, da somit der Code besser dokumentiert werden kann

Vue.js Components

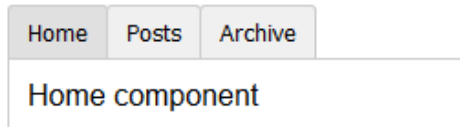
Dynamische Components

- Bisher haben wir Components nur statisch an eine Webseite angebunden, es kann aber auch Fälle geben, in denen Components dynamisch ausgetauscht werden sollen
- Hierfür stellen Vue Components das Spezialattribut `is` zur Verfügung

```
<component v-bind:is="dynamischeComponent"></component>
```
- `dynamischeComponent` kann hierbei einen von zwei Werten enthalten
 - Den Namen der registrierten Component (bei Globalen Components)
 - Das Component Objekt aus der component Option (bei Lokalen Components)

Vue.js Components

Dynamische Components



- Wenn eine Component mithilfe des **is Attributes** ausgetauscht wird, wird die gesamte Component neu erstellt
- Betrachten wir folgendes Beispiel eines Interfaces, das zwischen verschiedenen Tabs wechselt
- Jedes mal , wenn zwischen den Tabs gewechselt wird, wird die gesamte Component neu erstellt
- Besser wäre hier den Status einer Component zu speichern um diese später nochmals nutzen zu können, ohne sie neu erstellen zu müssen

Vue.js Components

Dynamische Components

- Um eine Component auf der Webseite zu „speichern“, kann eine Component in ein `<keep-alive>` Tag eingeschlossen werden

```
<keep-alive>  
  <component v-bind:is="dynamischeComponent"></component>  
</keep-alive>
```

- Durch das `<keep-alive>` Tag, wird der Status der Component beibehalten, auch wenn diese nicht aktiv ist

Vue.js Components

HTML Parsing Probleme

- Einige HTML Elemente haben Restriktionen, welche Elemente innerhalb dieser Elemente genutzt werden z.B. `<table>`, `` oder ``
- In diesen Elementen werden die Element `<tr>` bzw. `` erwartet

```
<table>  
  <component-a></component-a>  
</table>
```

Da `component-a` von dem HTML Parser nicht als `<tr>` erkannt wird, wird ein Fehler ausgeworfen

- Dieses Problem kann mithilfe des `is Attributes` umgangen werden

```
<table>  
  <tr is="component-a"></tr>  
</table>
```