Bouw zelf een Tesla Battery Range Calculator

Het kan met Vue.js!

Deze tutorial gaat helemaal in op het nieuwe autorijden. Dat elektrisch rijden de toekomst heeft, is onderhand wel duidelijk. Maar hoe ver kun je nou eigenlijk rijden op een volle accu? En welke invloed heeft bijvoorbeeld de snelheid, buitentemperatuur en de grootte van de velgen op het bereik? In deze tutorial gaan we aan de slag met Vue.js: een gemakkelijk te begrijpen JavaScript framework. We gaan een dashboard maken, waarmee berekend kan worden hoeveel range de Tesla heeft onder verschillende omstandigheden.

Als uitgangspunt voor de tutorial, clone je deze Github repository: **https://github.com/petereijgermans11/workshop-reactjs-vuejs/**

```
cd workshop-reactjs-vuejs/vuejs-app
```

Lees de **README.md** voor de uit te voeren opdrachten. Afbeelding 1 is een voorbeeld van de applicatie, die we gaan bouwen. We gaan initieel uit van een buggy applicatie, die we moeten fixen en verder uitbouwen. Voordat we aan de slag gaan, leg ik eerst uit hoe deze applicatie is opgebouwd.

Requirements

Om te starten met deze tutorial installeer je het volgende:

- stable node version 8.9 of hoger (**https://nodejs.org/en/download/**)
- yarn (**https://yarnpkg.com**)

Project structuur

Het project waaraan we werken (zie Listing 1) heeft dezelfde structuur als Afbeelding 1 en geeft de componenten weer waaruit deze applicatie bestaat. De **main.js** is de entry point van de applicatie. **App.vue** is de entry component van de applicatie. De buitenste rand van Afbeelding 1 geeft de App component weer.

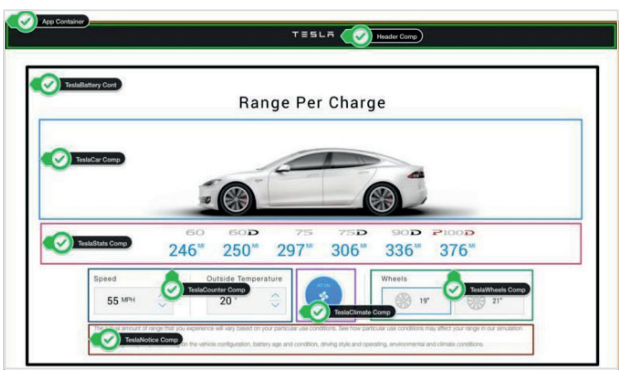
Project entry point

Een Vue applicatie wordt opgestart in **main.js** (zie Listing 2). In deze main.js maak je eerst een nieuwe "root Vue instantie" aan. Dit gaat als volgt:

1. het importeren van Vue: `import Vue from 'vue'`
2. het importeren van een entry component App.vue, middels: `import App from './App.vue'`
3. en het aanmaken van een `'root Vue instance': new Vue({….})`

```
vuejs-app
  public
  src
    assets
    tesla-battery
      components
        tesla-car.component.vue
        tesla-climate.component.vue
        tesla-counter.component.vue
        tesla-stats.component.vue
        tesla-wheels.component.vue
      tesla-battery.component.vue
      tesla-battery.service.js
      tesla-header.component.vue
    App.vue
    main.js
    styles.css
    tesla-style.scss
  package-lock.json
  package.json
  README.md
  yarn.lock
```

Listing 1



Afbeelding 1

Peter Eijgermans is Full Stack Developer en Codesmith bij Ordina JTech.7 Hij deelt graag zijn kennis met anderen middels workshops en presentaties.

```
import Vue from 'vue';
import App from './App.vue';

new Vue({
  render: h => h(App),
}).$mount('#app');
```

Listing 2

4. vanuit deze 'root Vue instantie render je de geïmporteerde App.vue component (entry component).

```
render: h => h (App)
```

5. tenslotte wordt deze root Vue instantie gemount. Dit is het punt waar de applicatie wordt opgestart. Hierbij wordt gerefereerd naar een HTML-element met als identificatie *#app*, die gedefinieerd is in de template in de App.vue component.

App.vue component

Deze *App.vue* (zie Listing 3) is de entry component van de applicatie en bestaat uit de volgende onderdelen:

script: dit is het JavaScript gedeelte van dit component. In bovenstaand voorbeeld geeft de *name property* de naam van de component aan (de naam is "`app`"). In de *components property* worden de child componenten gedefinieerd, die dit component gebruikt. In dit geval is `TeslaBattery` een child component van de App.vue component. Om gebruik te maken van de `TeslaBattery` component moet deze eerst geïmporteerd worden (`import TeslaBattery from '…'`). In de *data()-function* kun je state variabelen definiëren en initialiseren, zoals het geïmporteerde logo en de greeting property. Voor het renderen van het logo en de greeting dienen deze in de template (zie Listing 3) gedefinieerd te worden. Dit gehele component moet je uiteindelijk exporteren (middels `export default`), zodat deze weer geïmporteerd kan worden in andere componenten en in main.js.
template: is verantwoordelijk voor het definiëren van de output, die de component genereert. Vue.js gebruikt een op HTML-gebaseerde template syntax. Data afkomstig uit de *data()-function* kan eenvoudig gerenderd worden door middel van *databinding*. De meest eenvoudige vorm van *databinding* is tekst interpolatie met behulp van de `Mustache` syntax (dubbele accolades): `{{greeting}}`

```
<template>
    <div id="app">
        <img :src="logo">

        {{ greeting}}

        <tesla-battery></tesla-battery>

    </div>
</template>

<script>
    import TeslaBattery from './tesla-battery/tesla-battery.component';

    import logo from './assets/logo.svg';

    export default {
        name: 'app',
        components: {
            TeslaBattery,
        },
        data() {
            return {
                logo,
                greeting: "Hello TESLA !!!"
            };
        }
    };
</script>

<style lang="scss">
    @import 'tesla-style.scss';
</style>
```

Listing 3

In bovenstaand voorbeeld wordt `{{greeting}}`, vervangen door de waarde `Hello Tesla !!!` uit de betreffende data-function. Boven deze greeting wordt ook nog het logo gerenderd dankzij de `img`-tag. Om het logo toe te wijzen aan de *img src-attribute*, dient gebruik gemaakt te worden van *attribute binding*. Hiervoor kan je `v-bind` of `:` gebruiken (`<img :src= "logo">` of `<img v-bind:src= "logo">`). Attribute binding wordt veelvuldig toegepast in deze applicatie.
En tenslotte wordt middels de `<tesla-battery>` - *tag* de TeslaBattery component geïnstantieerd en gerenderd. Voor deze tag (ook wel "custom element" genoemd) dien je de Kebab-case te gebruiken. Hoe dit component functioneert zal verderop behandeld worden.
style: in Vue gebruiken we een SCSS-file voor het stylen van de gehele applicatie.

Breaking Down the UI

Bijna alle Vue-applicaties bestaan uit een samenstelling van componenten. Deze applicatie bestaat uit een entry App-component met de TeslaBattery als child component. En de TeslaBattery component bevat de volgende child componenten:
TeslaCar: voor het renderen van de *TeslaCar* image met wielanimatie.
TeslaStats: voor het renderen van de **maximum battery range per Tesla-model.** Het gaat hier om de modellen: 60, 60D, 75, 75D,90D en P100D.

BIJNA ALLE VUE-APPLICATIES BESTAAN UIT EEN SAMENSTELLING VAN COMPONENTEN

TeslaCounter: voor het handmatig regelen van de snelheid (speed) en de buitentemperatuur.
TeslaClimate: deze verandert de verwarming naar airco wanneer de buitentemperatuur meer dan 20 graden is.
TeslaWheels: voor het handmatig aanpassen van de wielmaat van 19 inch naar 20 inch en vice versa.
De gebruikersinterface wordt als volgt weergegeven door een componenten tree (zie Listing 4).

```
<App>                              <!-- Application entry point -->
    <TeslaHeader></TeslaHeader>
    <TeslaBattery>                 <!-- Container component -->
        <TeslaCar/>                <!-- Presentational component -->
        <TeslaStats/>              <!-- Presentational component -->
        <TeslaCounter/>            <!-- Presentational component -->
        <TeslaClimate/>            <!-- Presentational component -->
        <TeslaWheels/>             <!-- Presentational component -->
        <TeslaNotice/>             <!-- Presentational component -->
    </TeslaBattery>
</App>
```

Listing 4

Uit Listing 4 blijkt dat de "Tesla Battery component" een *Container componen*t is. Hierbij zijn de onderliggende child componenten *Presentation componenten*. Dit is een handig patroon dat kan worden gebruikt bij het ontwikkelen van een Vue-applicatie. Door componenten in twee categorieën te verdelen, zijn ze beter herbruikbaar.

Container componenten kenmerken zich door het volgende:
- ze kunnen zowel presentatie- als container componenten bevatten;
- ze zorgen voor het aanmaken en doorgeven van data aan child componenten middels "props";
- ze voeren logica uit op basis van binnengekomen events;
- ze zijn verantwoordelijk voor het managen van de state en weten wanneer een component opnieuw moet worden gerenderd;
- ze zijn vaak stateful, omdat ze de neiging hebben om als gegevensbronnen te dienen.

Presentation componenten kenmerken zich door het volgende:
- ze worden ook wel "dumb components" genoemd. De focus ligt op de gebruikersinterface. Vrijwel alle basis UI-componenten moeten als dumb components worden beschouwd. Voorbeelden hiervan zijn buttons, inputs, modals etcetera. In ons voorbeeld is de TeslaCar ook een dumb component, die zorgt voor het renderen van de TeslaCar image;
- ze ontvangen data via "props" en geven data terug aan parent componenten door middel van een event;
- ze zijn vaak stateless en hebben geen afhankelijkheid met de rest van de applicatie.

Deze aanpak heeft de volgende voordelen:
- herbruikbaarheid;
- dumb components zijn gemakkelijker te testen, omdat ze enkel "props" ontvangen, events emitten en een stukje UI retourneren;
- hogere leesbaarheid: hoe minder code je hebt en hoe beter het georganiseerd is, des te makkelijker is het te begrijpen en aan te passen;
- het biedt consistentie en voorkomt duplicatie van code.

TeslaBattery service

De gegevens die we gebruiken, zijn hard-coded vastgelegd in **tesla-battery.service.js**. Deze service heeft een methode `getModelData()` voor het ophalen van de modeldata. Bekijk de structuur van deze modeldata in Listing 5.

```
export default {
  getModelData() {
    return {
      '60': {
        19: {
          on: {
            speed: {
              45: {
                '-10': 224,
                '0': 255,
                '10': 287,
                '20': 289,
                '30': 287,
                '40': 258,
              },
              50: {
                '-10': 211,
                '0': 238,
                '10': 264,
                '20': 267,
                '30': 267,
                '40': 244,
              },
```

Listing 5

Op basis van de volgende parameters wordt de **maximum battery range** per Tesla-model bepaald: Tesla-model (60, 60D...), wheelsize (19/20 inch), climate (on/off), speed en temparature (-10, 0 ...).

TeslaBattery component

Dit component is verantwoordelijk voor het definiëren, aanmaken en doorgeven van data aan child componenten middels "props". Tevens is het verantwoordelijk voor het managen van de state van de applicatie.

Helemaal ingeklapt (zie Listing 6) zien we dat dit component bestaat uit onderstaande properties.

```
export default {
  name: 'tesla-battery',
  components: {…
  },
  data() {…
  },
  computed: {…
  },
  methods: {…
  },
};
</script>
```

Listing 6

De `components` property bevat alle child componenten waarvan dit component gebruik maakt. De `computed` property bevat de functies, die gecached worden. Dat wil zeggen dat zo een functie alleen uitgevoerd wordt als deze afhankelijk is van een specifieke data property en wanneer de state van deze property wijzigt. In onderstaande volledige versie van de `TeslaBattery` component (zie Listing 8), is de `stats()-functie` een voorbeeld van een `computed` function. Deze functie filtert de "maximum battery range per Tesla-model" uit de *modeldata* (zie **Listing 5**). In Listing 7 is een voorbeeld van de output van de stats()-functie. Deze maximum battery range is gebaseerd op de gebruikersinvoer, zoals de geselecteerde `wheelsize`, `climate`, `speed` en `temperature`. En deze stats()-functie wordt alleen uitgevoerd als deze gebruikersinvoer wijzigt. De gebruikersinvoer wordt vastgelegd in het tesla-object (state-object), die gedefinieerd is in de *data-function*.

```
Maximum battery range per model:
[
  {"model":"60","miles":267},
  {"model":"60D","miles":271},
  {"model":"75","miles":323},
  {"model":"75D","miles":332},
  {"model":"90D","miles":365},
  {"model":"P100D","miles":409}
]
```

Listing 7

De `methods` property bevat de alle functies die **niet** gecached worden. De `changeClimate()` functie wordt hierin gedefinieerd, omdat deze functie getriggerd wordt door een onClick event (en niet op basis van een data/state property) (zie listing 8).

```
<template>
  <form class="tesla-battery">
    <tesla-car :wheelsize="tesla.wheels" :speed="tesla.speed" />
    <tesla-stats :stats="stats" />
      <tesla-counter title="Speed" unit="kmh" :step="5" :min="45" :max="70" v-model="tesla.speed" />
        <tesla-counter title="Outside Temperature" unit="°" :step="10" :min="-10" :max="40" v-model="tesla.temperature" />
        <tesla-climate :limit="tesla.temperature > 10" :value="tesla.climate" :onClick="changeClimate" />
      <tesla-wheels v-model="tesla.wheels" />
  </form>
</template>
```

Listing 9

De template van deze *TeslaBattery* component heeft dezelfde structuur als Listing 4, namelijk (zie Listing 9).

Doorgeven van data aan child componenten middels props

In Afbeelding 2 wordt `stats` data (afkomstig uit de stats()-functie, zie Listing 7) vanuit de `TeslaBattery` component aan de `TeslaStats` component doorgegeven.
Om data door te geven aan een child component, dien je `v-bind` of `:` te gebruiken in de template van de `TeslaBattery` component (zie Listing 10).

```
<template>
   <form>
      ...
      <tesla-stats :stats="stats" />
      ...
   </form>
</template>
```

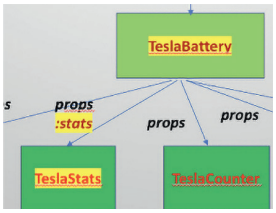
Listing 10

Ontvangen van data in de TeslaStats Component

Dit component bevat een `props` property in de scripts-sectie, voor het ontvangen van de `stats` data. Deze `stats` is van het type *Array* (zie Listing 11). In de template wordt door de "stats" geïtereerd, met behulp van een `v-for` directive van Vue.js. De `:key` geeft aan dat deze lijst in een bepaalde volgorde moet worden gerenderd.
In de `filter` property kun je een custom filter definiëren. In de template wordt een custom filter "*lowercase*" met een pipe `|` aangeroepen voor de te renderen modelnamen. Er is ook een custom filter gedefinieerd voor het omzetten van miles naar "`km`".

Tenslotte

Met deze introductie kun je beginnen aan het oplossen van bugs en het uitvoeren van opdrachten, die beschreven zijn in de README.md van dit project. In dit Github project is ook een Powerpoint toegevoegd, die dieper ingaat op zaken als two-way databinding middels de `v-model` directive, toewijzen van een onClick event aan een button met `@click` en het aanmaken van andere componenten. ■



Afbeelding 2

```
<script>
import TeslaCar from './components/tesla-car.component';
import TeslaClimate from './components/tesla-climate.component';
import TeslaCounter from './components/tesla-counter.component';
import TeslaStats from './components/tesla-stats.component';
import TeslaWheels from './components/tesla-wheels.component';

import teslaService from './tesla-battery.service';

export default {
  name: 'tesla-battery',
  components: {
    TeslaCar,
    TeslaClimate,
    TeslaCounter,
    TeslaStats,
    TeslaWheels,
  },
  data() {
    return {
      color: "#0000ff",
      title: 'Ranger Per Charge',
      results: ['60', '60D', '75', '75D', '90D', 'P100D'],
      tesla: {
        speed: 55,
        temperature: 20,
        climate: true,
        wheels: 19,
      },
    };
  },
  computed: {
    models() {
      return teslaService.getModelData();
    },
    stats() {
      return this.results.map(model => {
        const {speed, temperature, climate, wheels} = this.tesla;
        const miles = this.models[model][wheels][climate ? 'on' : 'off'].speed[
          speed
        ][temperature];
        return {
          model,
          miles,
        };
      });
    },
  },
  methods: {
    changeClimate() {
      this.tesla.climate = !this.tesla.climate;
    },
  },
};
</script>
```

Listing 8

```
<template>
  <div class="tesla-stats">
    <ul>
      <li v-for="stat in stats" :key="stat.model">
        <div :class="'tesla-stats-icon tesla-stats-icon--'+stat.model |  lowercase"></div>
        <p>{{ stat.miles | km }}</p>
      </li>
    </ul>
  </div>
</template>

<script>
export default {
  name: 'tesla-stats',
  props: {
    stats: {
      type: Array,
      required: true,
    },
  },
  filters: {
    lowercase(value) {
      return !value ? '' : value.toLowerCase();
    },
    km(value) {
      return Math.floor(value * 1.609344);
    },
  },
};
</script>
```

Listing 11