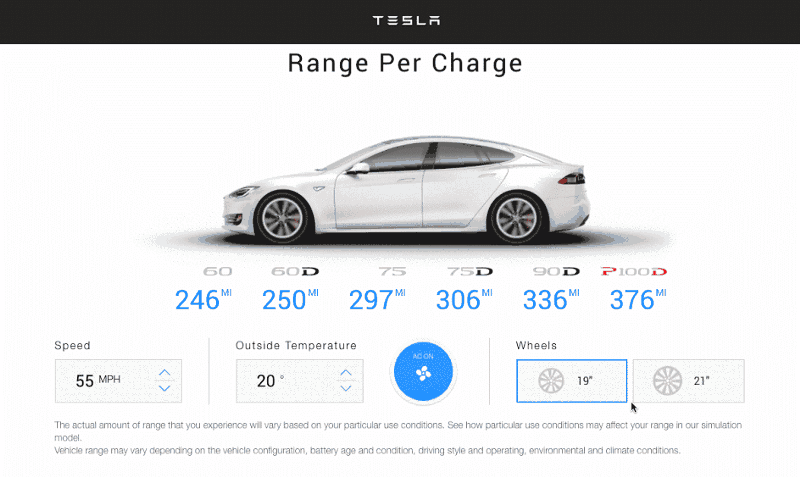
**Building Tesla’s Battery Range Calculator with Vue**

In dit artikel zal ik je helpen met het bouwen van een Tesla **Battery Range Calculator** met Vue.

Dit is de GIF-afbeelding van onze applicatie, die we gaan bouwen.



Laten we nu de applicatie stap voor stap gaan maken.

**Project Setup and create-Vue-app**

**Requirements**

* Installeer de stable **node** version 8.9 of hoger **(**<https://nodejs.org/en/download/>**)**
* Install **yarn** ([https://yarnpkg.com](https://yarnpkg.com/))
* Installeer de **vue CLI**, de standaard tooling om Vue.js te ontwikkelen:

npm install -g @vue/cli

*# OR*

yarn global add @vue/cli

Je kan vervolgens de juiste versie checken (**3.x**) met dit commando:

**vue --version**

**Creating a project**

Vue create

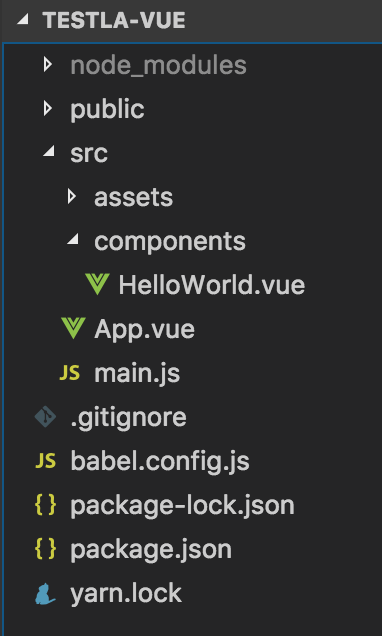
Om een project aan te maken en op te starten, run het volgende in je terminal (voor windows omgevingen verwijs ik je naar <https://cli.vuejs.org/guide>):

* vue create **tesla-vue**
* cd testla-vue
* yarn serve

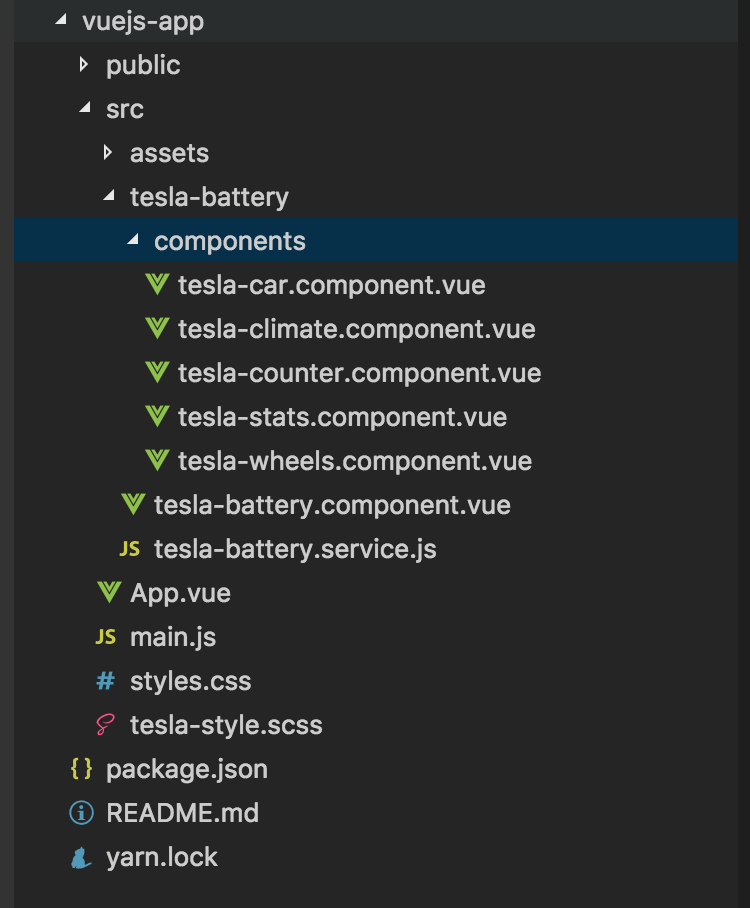
Vervolgens wordt op <http://localhost:8080> deze pagina geopend:



Het huidige project heeft de volgende structuur:



En het project zal uiteindelijk deze structuur moeten krijgen:



Het uiteindelijke resultaat kan je hier downloaden: <https://github.com/petereijgermans11/workshop-reactjs-vuejs/>

**Project entry point**

Eerst moeten we de entry points instellen om onze Tesla-app te starten. Gelukkig zijn deze al gemaakt:

* **src/main.js**
* **src/App.vue**

Een Vue applicatie wordt gebootstrapped in je main.js. In deze main.js maak je eerst een nieuwe ‘root Vue instance’ aan. Dit gaat als volgt:

1. het importeren van vue, middels:

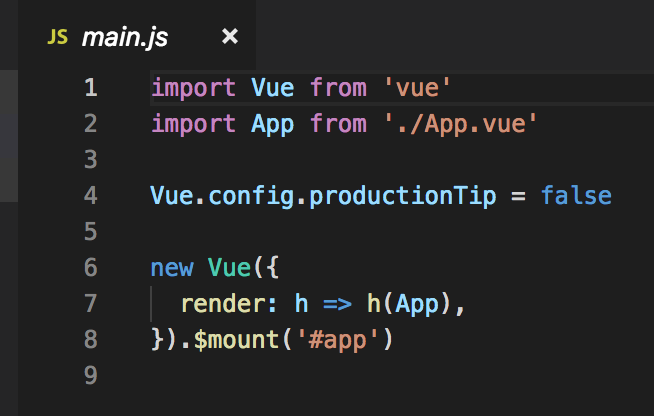
**import Vue from ‘vue’**

2. het importeren van je bestaande entry component App.vue, middels:

**import App from ‘./App.vue’**

3. en het aanmaken van een ‘root Vue instance’, middels:

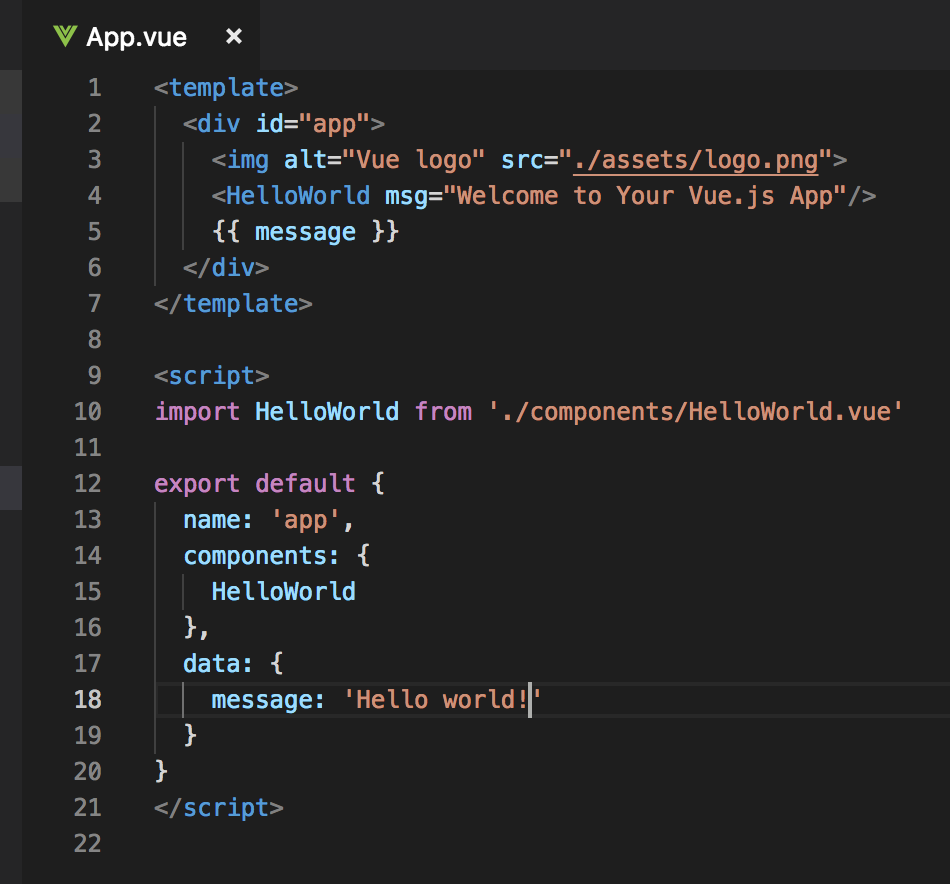
**new** Vue({….})



4. vanuit deze ‘root Vue instance’ render je de geïmporteerde App.vue component (entry component)

render: h => h (App)

5. en tenslotte wordt deze root Vue instance ‘ge-mount’. Dit is het punt waar de applicatie wordt gebootstrapped. Hierbij wordt gerefereerd naar je #app-id afkomstig uit de template in je App.vue component.



Dit App.vue component bestaat uit de volgende onderdelen:

* **template:** Vue.js gebruikt een op HTML gebaseerde template syntax waarmee je de gegenereerde DOM declaratief kunt binden aan de gegevens van de onderliggende Vue-instantie. De meest eenvoudige vorm van databinding is tekst interpolatie met behulp van de “Mustache” syntax (dubbele accolades): {{message}}

In bovenstaand voorbeeld wordt {{message}}, vervangen door de waarde message property uit het betreffende data-object.

* **script**: Dit gedeelte bevat de naam, de logica en de state van het betreffende component.

In bovenstaand voorbeeld geeft de *name-sectie,* de naam van het component aan.

In de *component-sectie* worden de child-componenten gedefinieerd. In dit geval is ‘HelloWorld’ een child-component van dit component. En in de *data-sectie* kan je state variabelen definiëren en initialiseren.

**Project images**

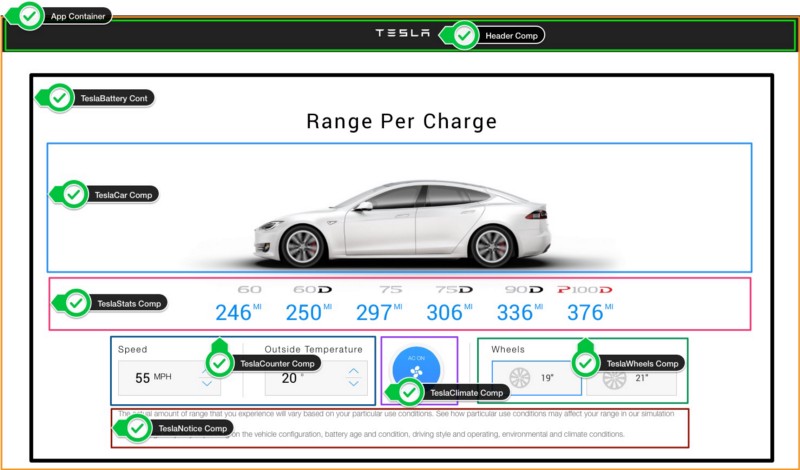
Alle benodigde project images kunnen vanaf hier gedownload worden:

<https://github.com/petereijgermans11/workshop-reactjs-vuejs/tree/master/workshop-reactjs-vuejs-solution/vuejs-app/src/assets>

**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 childcomponent. En het TeslaBattery component bevat volgende childcomponenten:

* TeslaCar: voor het renderen van de TeslaCar image met wiel animatie
* TeslaStats: voor het renderen van de **maximum battery range per Tesla-model**
* TeslaCounter: voor het handmatig regelen van de snelheid (speed) en de buiten temperatuur
* TeslaClimate: deze veranderd de verwarming in airco wanneer deze meer dan 20 graden is, en verwarming wanneer deze lager is dan 20 graden
* TeslaWheels: voor het handmatig aanpassen van de wiel maat van 19 inch naar 20 inch en visa versa.



De gebruikersinterface wordt als volgt weergegeven door een componenten tree:

<**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**>

Hieruit blijkt dat de ‘Tesla Battery component’ een Container component is. Dat houdt in dat dit component zorgt voor het aanmaken en managen van data/state en voor de logica binnen de applicatie. Dit soort componenten zijn statefull.

Presentational componenten zijn stateless en presenteren uitsluitend data.

Waarom deze benadering zou je zeggen?

Ten eerste Better separation of concerns. En herbruikbaarheid van componenten en daardoor ook gemakkelijker te testen.

**Header component**