**Kennissessie**

**TypeScript**

**Vue met TypeScipt**

**Feb 2020**

**Auteur: Arno McLean**

**Installation Guide to follow lessons:**

1. Install Google Chrome
2. Install VisualStudio Code: <https://code.visualstudio.com/Download>
3. Install Git Als je visual studio op je laptop hebt staan dan heb je GIT er al opstaan. Mocht dat niet zo zijn dan doe het volgende: <https://git-scm.com/download/win> Je kunt checken of je Git hebt door command line te openen en dan Git in de tikken en Enter. Als je git hebt krijg je een help file te zien. Andere installeer hem door na het bovenste te gaan.
4. Install Node: <https://nodejs.org/en/download/>
5. Install Typescript:
   1. Open up command prompt
   2. Type in: npm install -g typescript
6. Install vue-cli:
   1. Open up command prompt
   2. Type in: npm install -g @vue/cli
7. Install plug-ins (Extensions) for Visual Studio code:
   1. Open up visual studio code
   2. Klik op Extensions in de side bar 
   3. Zoek en instaleer de volgende Plug-ins:
      1. Vue (by jcbuisson)
      2. Vetur
8. Install Vue Plug in for Chrome: <https://chrome.google.com/webstore/detail/vuejs-devtools/nhdogjmejiglipccpnnnanhbledajbpd>

Inleiding met TypeScript

**Introduction**

Waarom Typescript, Javascript werkt toch ook? Nou ja dat klopt ook wel, maar zie Typescript iets wat je kan helpen om beter code te maken als je met de front-end werkt. Zie het ook als een manier om je front-end beter te laten werken met je backend. Zeker om dat je strongly typed werkt. Toen wij Vue gingen introduceren toen dacht ik ook van waarom waarom. En als je een tijdje er mee werkt dan valt het allemaal wel mee.

Dus zoals je al geraden hebt het gene wat Typescript je mee helpt is dat je Strongly typed gaat werken met Javascript. Het is een laagje dat boven op je Javascript zit. Als je je typescript hebt getyped dan gebruik je een pre-compilere die je code transformeerd na Javascript. Ik wil dus ook vandaag heel nadrukkelijk hier mee gaan werken, en via de commandline dit soort dingen gaan doen. Vandaar dat jullie gevraagt is om commandline tools te instaleren.

Typescript is gemaakt door mensen in Microsoft en daarna opensource gemaakt, het wordt dus ook wel breed gedragen door de community. Ik wil vandaag dus makkelijk ook een paar dingen gaan proberen en uitleggen aan jullie. Met wat voorbeelden gaan we wat programeren in TypeScript. En hopelijk hiermee krijgen jullie een gevoel wat je Typescript doet. Het is geen complete beeld er zullen best dingen missen, maar als we met Vue aan de gang gaan dan zul je hier veel dingen van terug zien.

Als we met Vue aan de slag gaan dan zullen we vue applicatie bouwen met Typescript, en deze optie kunnen we selecteren als je je Vue applicatie template bouwt met de Vue Command Line interface.

**Types en Javascript**

As I have mentioned before Typescript is all about Types, Javascript already has types (bool, string, number), but Typescript especially in the settings we currently have, tells you to specify what types variables are. It also introduces a Type called any, and we try to use this as little as possible. Saying that there are circumstances where we use this, but we will get back to this when we tackle Vue.

Javascript has types:

* String
* Number
* Boolean
* Null
* Undefined
* Object

Javascript is Dynamic which can be a good thing, but can also cause problems:

var aBoolean = false;

console.log(typeof aBoolean); // "boolean"

aBoolean = "Tom";

console.log(typeof aBoolean); // "string"

Here I have changed the code from a bool to a string.

Type script heeft dezelfde types, de belangrijkste die je daar aan toevoegd is eigenlijk ANY.

**Lesson 1**

In command line do the following:

* Mkdir typescript-tutorial
* Cd typescript-tutorial
* npm init -y
* npm i typescript --save-dev
* Open package.json and add to script the following line:

"tsc": "tsc"

* Then go back to command line and do the following: npm run tsc -- --init
* Open up tsconfig.json and replace the contents with this:

{

  "compilerOptions": {

    "target": "es5",

    "strict": true

  }

}

**Target** means which version of Javascript we are aiming the code to be compiled into.

**Strict** means how strict are Typescript rules applied to our code.

Oke, hoe kunnen we een programma schrijven in Typescript en deze via de command line iets laten zien.

* Create file called filterByTerm.tsfilterByTerm.ts
* Schrijf het volgende in de file:
* console.log(“hello world”);

Dan in je console:

1. Npm run tsc //Dit is de pre-compiler die de JS maakt
2. Node filterByTerm.js //Node runt de applicatie voor ons

Oke nou heb je dus een beetje javascript geschreven en node heeft het uitgevoerd.

**Laten we onze eerste programma gaan schrijven**

Add in this code:

function filterByTerm(input, searchTerm) {

  if (!searchTerm) throw Error("searchTerm cannot be empty");

  if (!input.length) throw Error("input cannot be empty");

  const regex = new RegExp(searchTerm, "i");

  return input.filter(function(arrayElement) {

    return arrayElement.url.match(regex);

  });

}

filterByTerm("input string", "java");

Je kunt nu al in Visual Studio code zien dat die niet gaat werken. Maar laten we ook eens zien wat de compiler zegt:

* Npm run tsc

As you can see we now need to fix this piece of code. What do you think is wrong?

Now let’s look at the code we wrote in javascript and fix it for typescript.

Fixes to script:

1. Add : string to the function
2. Make input to a string[] in function
3. Change calling function to first an array of string

**Lesson 2**

**Interfaces**

Voeg interface toe, dan krijg code werkende met dat.

interface Link {

  url: string;

}

Then explain about how arrayElement.url would actually better in an object with an interface.

**Lesson 3**

Laten we nu de Interface gaan uitbreiden:

Change Interface to reflect, explain optional items:

interface Link {

   url: string;

   id: number;

}

Then add in a ? 🡪 This means the field is allowed to be null, when creating instance you do not need to fill this field in the object.

interface Link {

   url: string;

   id?: number;

}

In VUE you will also see the !, this is what they call a non null assertion operator. Wat je eigenlijk zegt is dat de variable nooit null is. Deze wordt in VUE veel gebruikt wanneer je uit de VUEX store wat uitleest, daar over meer later.

**Lesson 4**

**Extend Inteface**

Created a new interface called TranslatedLink:

interface TranslatedLink extends Link {

   langugage: string;

}

Then add a new object to the array, in a seperate variable and add that to the array that you pass to the method.

Add an Extra console.log that get’s the translated Link

**Lesson5**

**Return Types**

Add : Link[] to the end of the function signature after the )

Then show how we can get the url from the return of the call

**Lesson6**

**Types aliases VS interfaces**

Explains types with in Typescript, interface is een vorm (shape) het is een vorm waarmee je aangeeft hoe je informatie er uit ziet. (in ons voorbeeld een Link). Type is een alias voor een custom type.

Voorbeeld:

type links = Array<Link>

const arrayOfLinks: links = [obj1, obj2, etc.. ]

Geef deze arrayOfLinks door aan de Method.

Show a push to an array

Wat is zijn:

* Const (constante waarde, zoals ook in C# bijvoorbeeld, je kan deze niet zomaar veranderen)
* Var (algemeen, gebruik hem niet)
* Let (local scope)

**Lesson 7**

**Interface and Objects**

1. New file interface-functions.ts
2. Add in this code:

interface Person {

  name: string;

  city: string;

  age: number;

}

const tom: Person = {

  name: "Tom",

  city: "Munich",

  age: 33,

  printDetails: function() {

    console.log(`${this.name} - ${this.city}`);

  }

};

1. Add in method name with a void return paramater and add in a call to tom.PrintDetails();
2. Add another method to your interface with parameters and a return type and print out via console.log and using the return of the new method

**Lesson 8**

**Referencing an object from another file**

With Import at the top of your file you can add in objects from another file or even another package. This is something we will be doing a lot in the upcoming Vue session.

1. Add export to the Person Interface in interface-function.ts
2. Add Import Person from to the filterByTerms and create an object in this file of type Person.
3. (or copy one from the other file)
4. Complie and show
5. Add in Export Default, exporting tom

Resource: <https://www.valentinog.com/blog/typescript/>

Vue

**Lesson 1: Create a Vue application using the CLI**

1. Open up Cmd / terminal
2. Create a directory you want to create you application in
3. Vue create basic
   * 1. Select manual, this allows us to create an application with all the feautres we need
     2. Select: Babel, TypeScript, Router, Vuex, CSS Pre-Processor, Linter
     3. Use class style component syntax
     4. Use babel / Yes
     5. Say no to history
     6. First Sass
     7. TsLint Standard Config
     8. Lint and fix on commit
     9. In dedicated Config files
     10. Save: Yes, choose template name
4. Open up Visual Studio code and have a look at the packages, compare to the Typescript one.
5. Start up application in Terminal inside Visual Studio code
6. Do a npm run build and show them the output
7. Go to website locahost:8080
8. Then go back into code to show application

**Lesson 2: de Applicatie**

1. Show main.ts
   1. Point out that application get’s rendered in the div with is app in Index.html
   2. Show how the router get’s imported as does the store. These are add on’s,
   3. Quickly explain the store (VUEX) and routing.

**Lesson 3: Separate the Html / TS in the Vue file**

1. Seperate vue into html / ts files:
   * 1. Add in this: npm install --save vue-template-loader webpack-stream
     2. Create shim file: shims-html.d.ts
     3. Add this code to it:

declare module '\*.html' {

    import Vue, { ComponentOptions, FunctionalComponentOptions } from 'vue'

    interface WithRender {

        <V extends Vue, U extends ComponentOptions<V> | FunctionalComponentOptions>(options: U): U

        <V extends typeof Vue>(component: V): V

    }

    const withRender: WithRender;npm ru

    export default withRender;

* + 1. Update tsconfig.json add: "src/\*\*/\*.html"
    2. Create file called: vue.config.js
    3. Add in code:

module.exports = {

  configureWebpack: {

    module: {

      rules: [

        {

          test: /.html$/,

          loader: "vue-template-loader",

          exclude: /index.html/

        }

      ]

    } } }

1. Then change the Component helloWorld.vue and move html into html
   1. Vue to ts remove CSS and Scripts tags
   2. Add in import WithRender from './HelloWorld.html'
   3. Add in @WithRender and @Component
   4. Change code in Home.vue, change import to HelloWolrd from HelloWorld.vue

**Lesson 4: Vuetify**

Adding Vuetify into project

* 1. npm install -g vuetify
  2. Npm add vuetify
  3. Change Main.ts
  4. Code:

import Vuetify from 'vuetify'

import 'vuetify/dist/vuetify.min.css'

Vue.config.productionTip = false

Vue.use(Vuetify)

const opts = {

 theme: { disable: false }

}

* 1. Add in New Vue: vuetify: new Vuetify(opts),
  2. Add <v-app> too app.vue

**Lesson 6: Counter and adding the view / component via the router**

1. Add a new compontent with Counter as H1, make this in a new folder
2. Explain router and add a route in Router and Add a link in App.Vue
3. Add 2 buttons and a text field
4. Add a public string in the component
5. Add a number
6. Write 2 methods that will add and minus the number and write them to the String
7. Hook string up to the TextField and hook methods to the button
8. Build in a rule: public fieldRule = [(v: string) => v !== "-1" || "We can not go below 0"]
9. and add it to :rules in the v-text-field.

**Lesson 7: Components**

1. Create a folder called child
2. Create file called clear.html and clear.ts
3. Add a button in clear.html
4. Add method that adds a click that emits an event this.$emit("clearField", "childComponent")
5. Add to the html where child components gets rendered @clearField=”clear” where clear is a new function in parent that takes a string.

**Lesson 8: Vuex**

1. Add in vuex-class and vuex-persist
   1. npm add --save vuex-persist
   2. npm add --save vuex-class
2. Create a new store, explain the methods that we will be using, Create interface and state of the store. Getters, Mutatans, Actions, Make sure when you create a store you set namespaced to true. Add store to the main store!
3. Create a new component that will consume the store.
4. Consume the store

**Lesson 5: Lint**

Add in lint config from CRM -CARe  add to: tslint.json

{

  "defaultSeverity": "error",

  "extends": [

    "tslint:recommended"

  ],

  "linterOptions": {

    "exclude": [

      "node\_modules/\*\*"

    ]

  },

  "rules": {

    "quotemark": [true, "double"],

    "indent": [true, "spaces", 2],

    "interface-name": false,

    "ordered-imports": false,

    "object-literal-sort-keys": false,

    "class-name": false,

    "no-debugger": true,

    "max-line-length": false,

    "no-shadowed-variable": false,

    "no-console": [true, "log", "debug", "info", "time", "timeEnd", "trace"],

    "member-ordering": false

  }

}

Run lint and run dev serve to fix lint problems

**Resource:**

Seperate files: <https://dev.to/georgehanson/building-vuejs-applications-with-typescript-1j2n>

Vuetify: <https://levelup.gitconnected.com/add-vuetify-to-your-vue-js-app-4e4e3616cb9c>