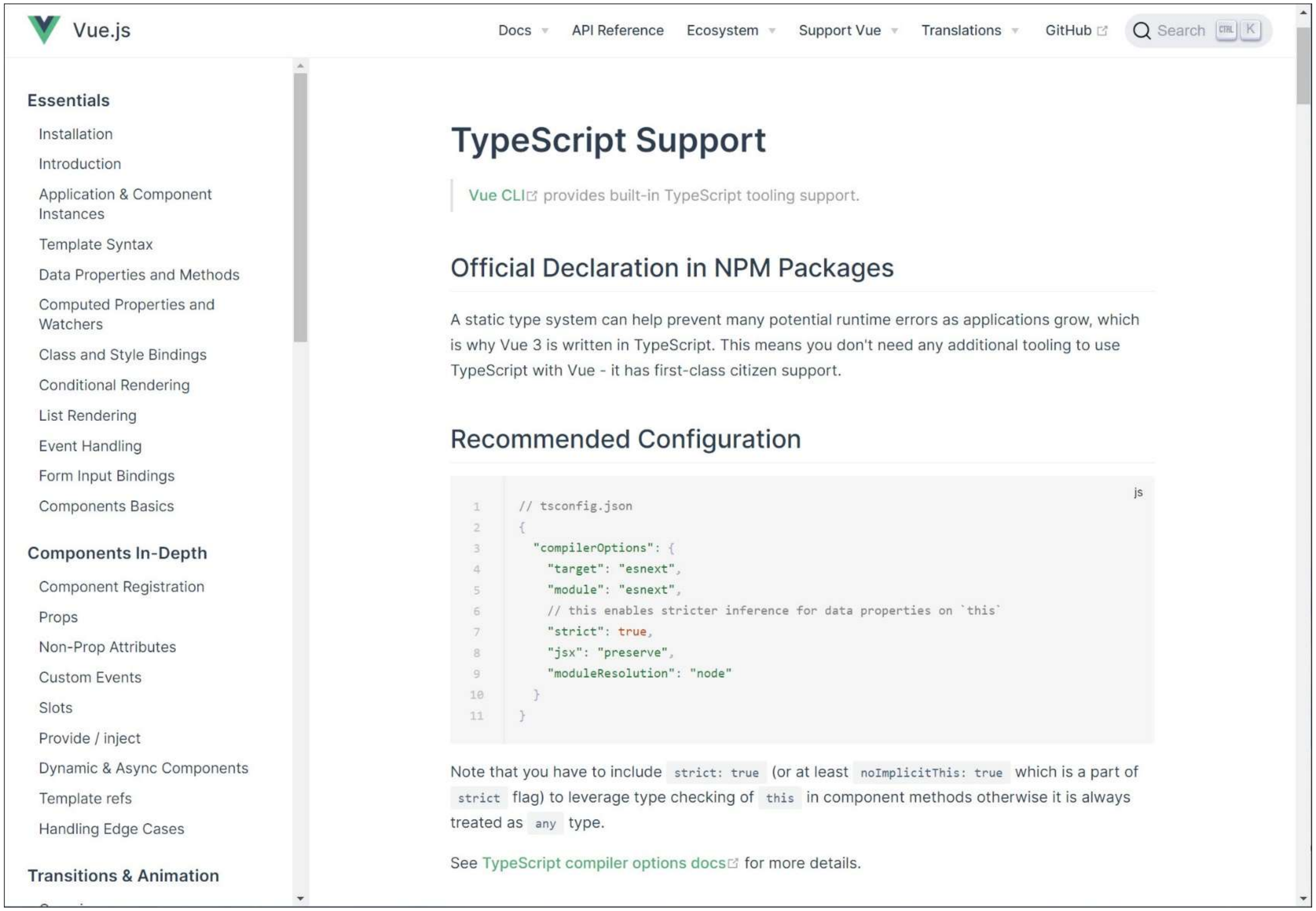
Vue Advanced Using TypeScript

Peter Kassenaar –
info@kassenaar.com

Official documentation



TypeScript Support

Vue CLI provides built-in TypeScript tooling support.

Official Declaration in NPM Packages

A static type system can help prevent many potential runtime errors as applications grow, which is why Vue 3 is written in TypeScript. This means you don't need any additional tooling to use TypeScript with Vue - it has first-class citizen support.

Recommended Configuration

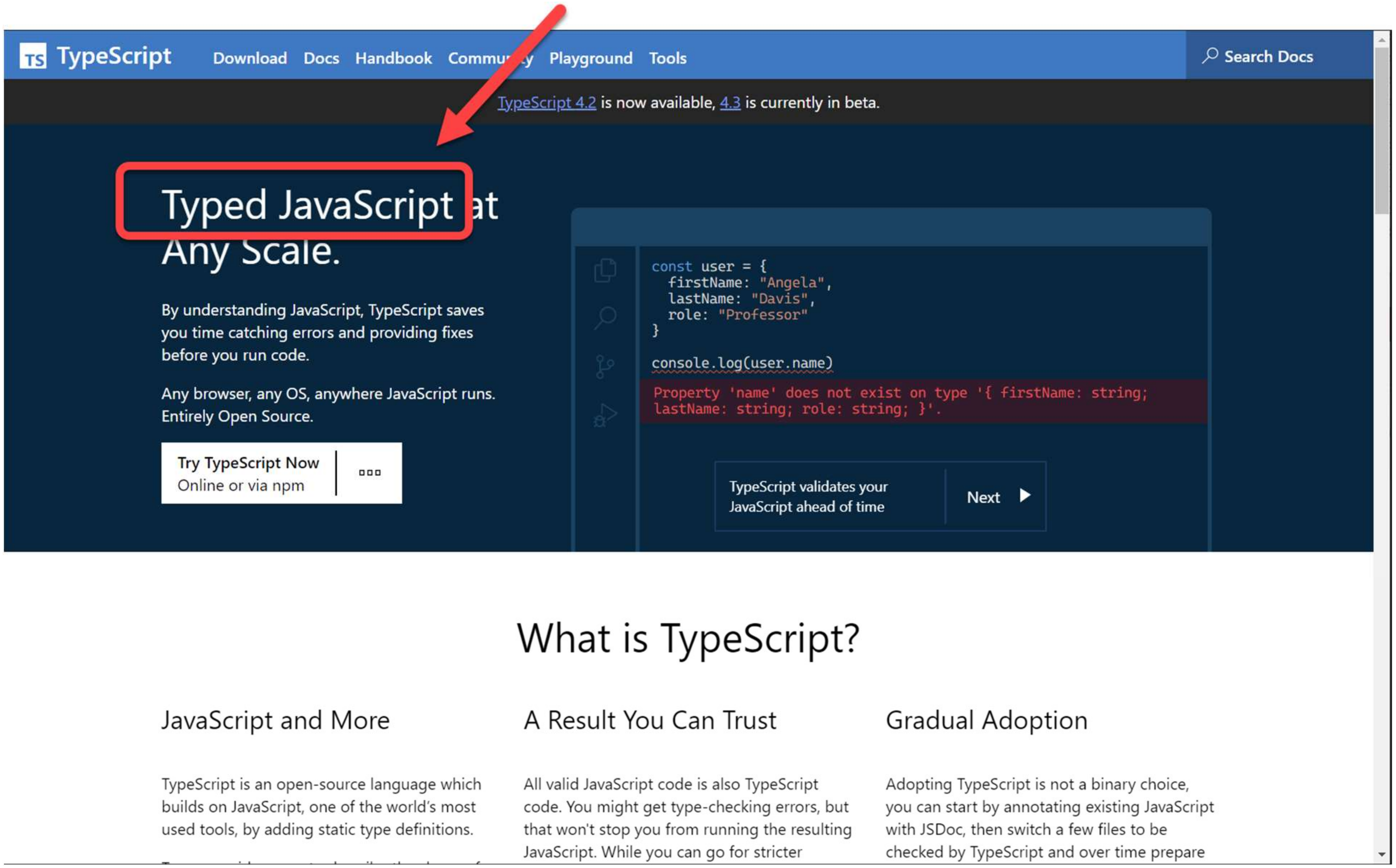
```js
// tsconfig.json
{
  "compilerOptions": {
    "target": "esnext",
    "module": "esnext",
    // this enables stricter inference for data properties on `this`
    "strict": true,
    "jsx": "preserve",
    "moduleResolution": "node"
  }
}
```

Note that you have to include `strict: true` (or at least `noImplicitThis: true` which is a part of `strict` flag) to leverage type checking of `this` in component methods otherwise it is always treated as `any` type.

See TypeScript compiler options docs for more details.

https://v3.vuejs.org/guide/typescript-support.html

A short primer on TypeScript



TypeScript 4.2 is now available, 4.3 is currently in beta.

Typed JavaScript at Any Scale.

By understanding JavaScript, TypeScript saves you time catching errors and providing fixes before you run code.

Any browser, any OS, anywhere JavaScript runs. Entirely Open Source.

```
const user = {
  firstName: "Angela",
  lastName: "Davis",
  role: "Professor"
}

console.log(user.name)
Property 'name' does not exist on type '{ firstName: string;
lastName: string; role: string; }'.
```

TypeScript validates your JavaScript ahead of time

What is TypeScript?

JavaScript and More

TypeScript is an open-source language which builds on JavaScript, one of the world's most used tools, by adding static type definitions.

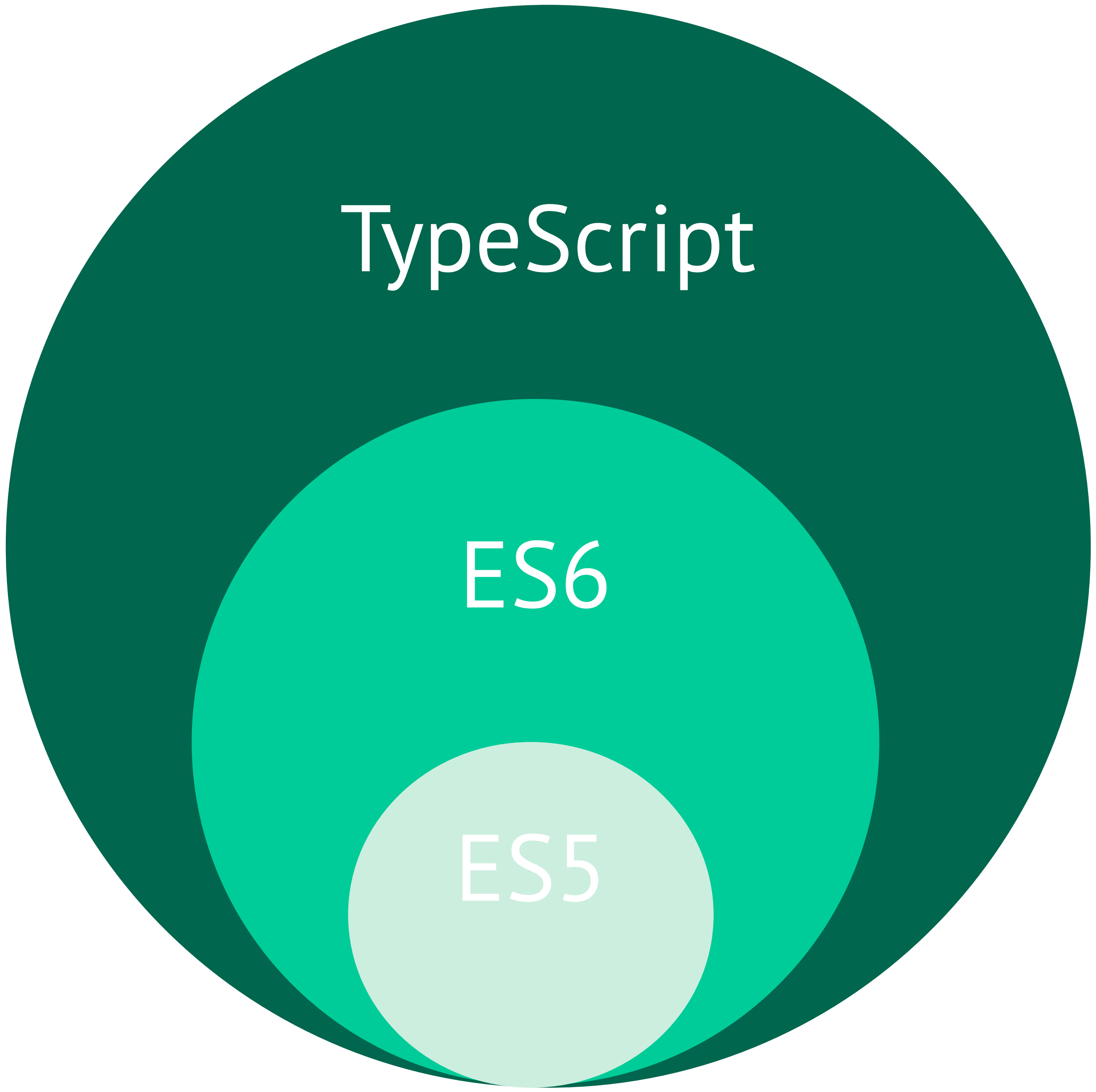
A Result You Can Trust

All valid JavaScript code is also TypeScript code. You might get type-checking errors, but that won't stop you from running the resulting JavaScript. While you can go for stricter

Gradual Adoption

Adopting TypeScript is not a binary choice, you can start by annotating existing JavaScript with JSDoc, then switch a few files to be checked by TypeScript and over time prepare

https://www.typescriptlang.org/


TypeScript
ES6
ES5

TypeScript and ECMAScript

Typescript is a superset of JavaScript

- It doesn’t replace JavaScript!
- TypeScript compiles to plain JavaScript, executed by the browser (or other application container)

TypeScript features:

- Annotations & types
- Interfaces, Generics, enums,
- Compiler, tooling support, and much, much more.

Everything in TypeScript is optional. You can always use just plain JavaScript.*

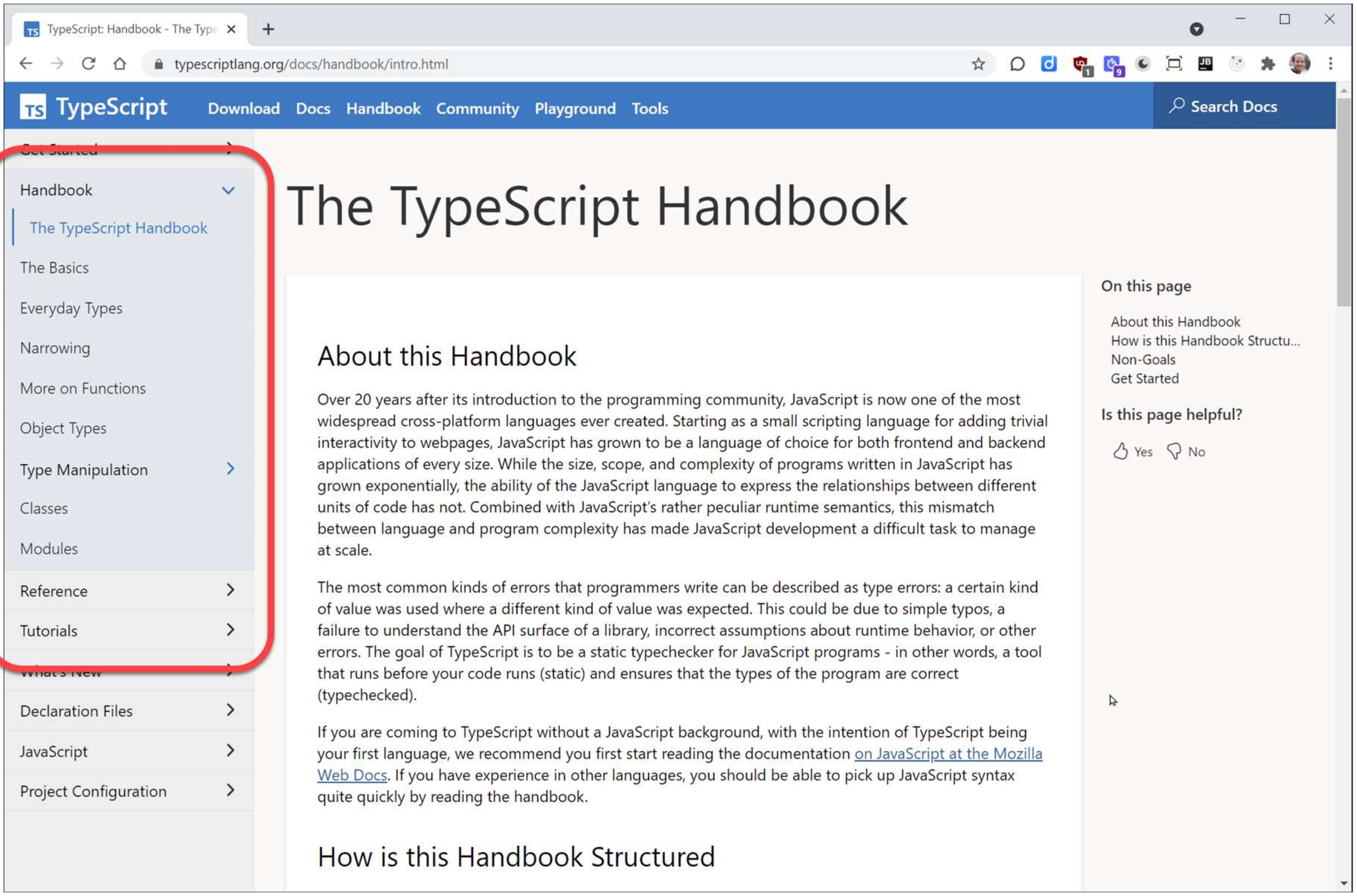
* Depending on the settings in your `tsconfig.json`

Using TypeScript in Vue

"A static type system can help prevent many potential runtime errors, especially as applications grow. "

https://v3.vuejs.org/guide/typescript-support.html

TypeScript Handbook



The TypeScript Handbook

About this Handbook

Over 20 years after its introduction to the programming community, JavaScript is now one of the most widespread cross-platform languages ever created. Starting as a small scripting language for adding trivial interactivity to webpages, JavaScript has grown to be a language of choice for both frontend and backend applications of every size. While the size, scope, and complexity of programs written in JavaScript has grown exponentially, the ability of the JavaScript language to express the relationships between different units of code has not. Combined with JavaScript's rather peculiar runtime semantics, this mismatch between language and program complexity has made JavaScript development a difficult task to manage at scale.

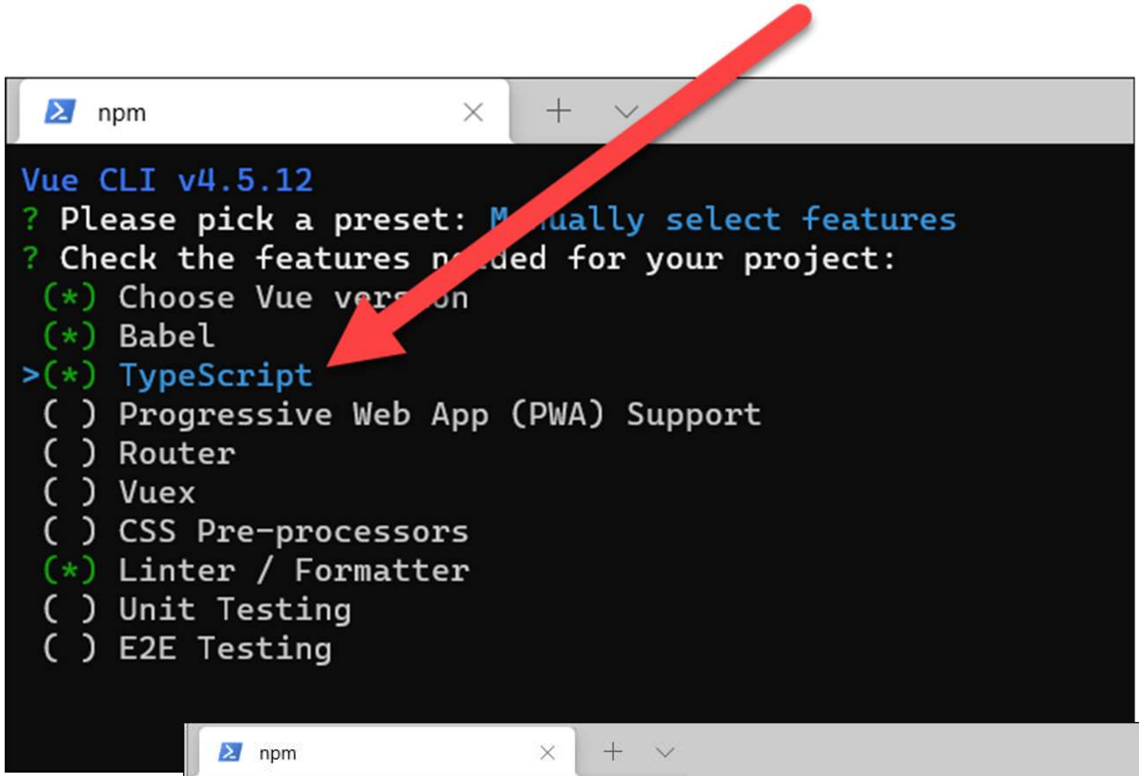
The most common kinds of errors that programmers write can be described as type errors: a certain kind of value was used where a different kind of value was expected. This could be due to simple typos, a failure to understand the API surface of a library, incorrect assumptions about runtime behavior, or other errors. The goal of TypeScript is to be a static typechecker for JavaScript programs - in other words, a tool that runs before your code runs (static) and ensures that the types of the program are correct (typechecked).

If you are coming to TypeScript without a JavaScript background, with the intention of TypeScript being your first language, we recommend you first start reading the documentation on JavaScript at the Mozilla Web Docs. If you have experience in other languages, you should be able to pick up JavaScript syntax quite quickly by reading the handbook.

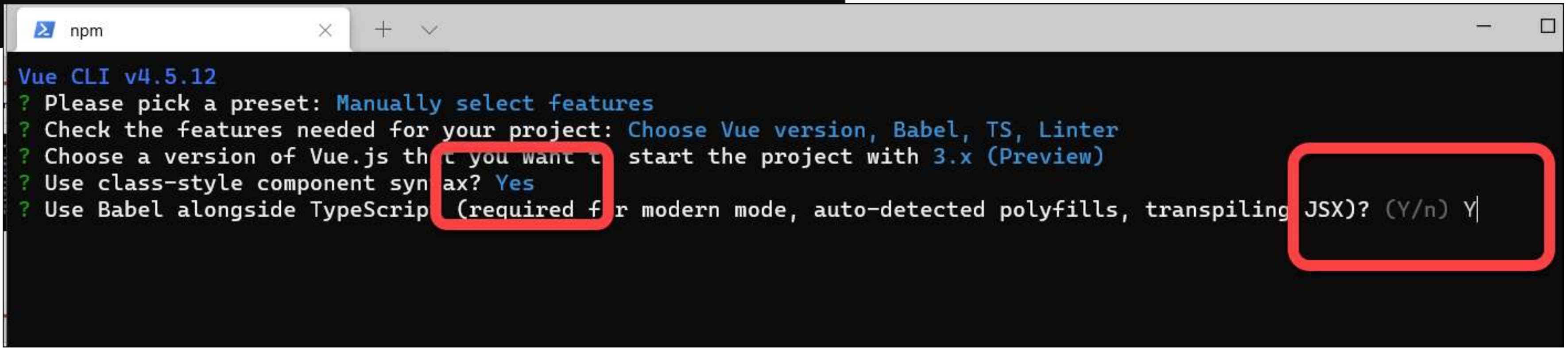
How is this Handbook Structured

https://www.typescriptlang.org/docs/handbook/intro.html

Using TypeScript with Vue CLI



```
Vue CLI v4.5.12
? Please pick a preset: Manually select features
? Check the features needed for your project:
 (*) Choose Vue version
 (*) Babel
>(*) TypeScript
 ( ) Progressive Web App (PWA) Support
 ( ) Router
 ( ) Vuex
 ( ) CSS Pre-processors
 (*) Linter / Formatter
 ( ) Unit Testing
 ( ) E2E Testing
```



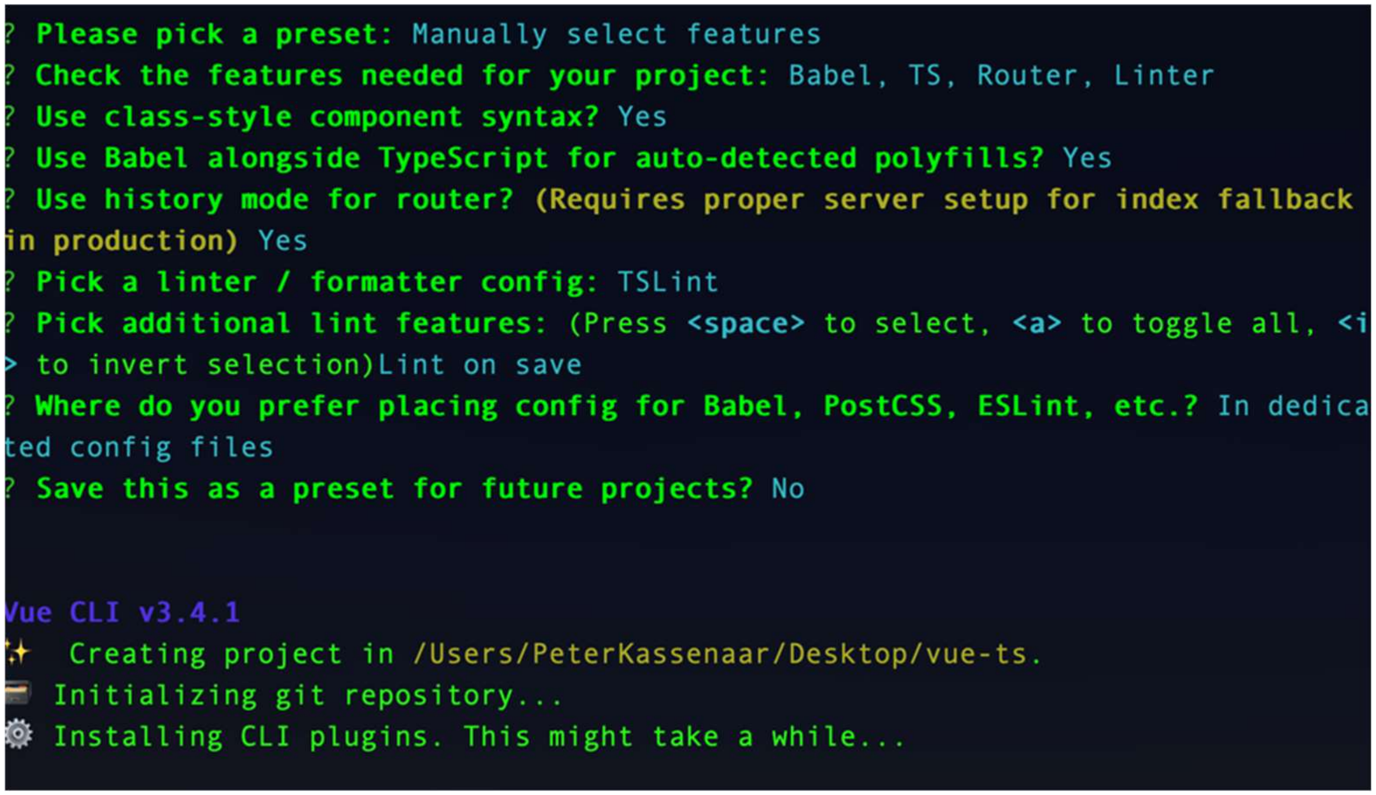
```
Vue CLI v4.5.12
? Please pick a preset: Manually select features
? Check the features needed for your project: Choose Vue version, Babel, TS, Linter
? Choose a version of Vue.js that you want to start the project with 3.x (Preview)
? Use class-style component syntax? Yes
? Use Babel alongside TypeScript (required for modern mode, auto-detected polyfills, transpiling JSX)? (Y/n) Y
```

Setting up a Vue/TypeScript project

Create a new project as usual, choosing `Manually select features`

Choose `TypeScript`

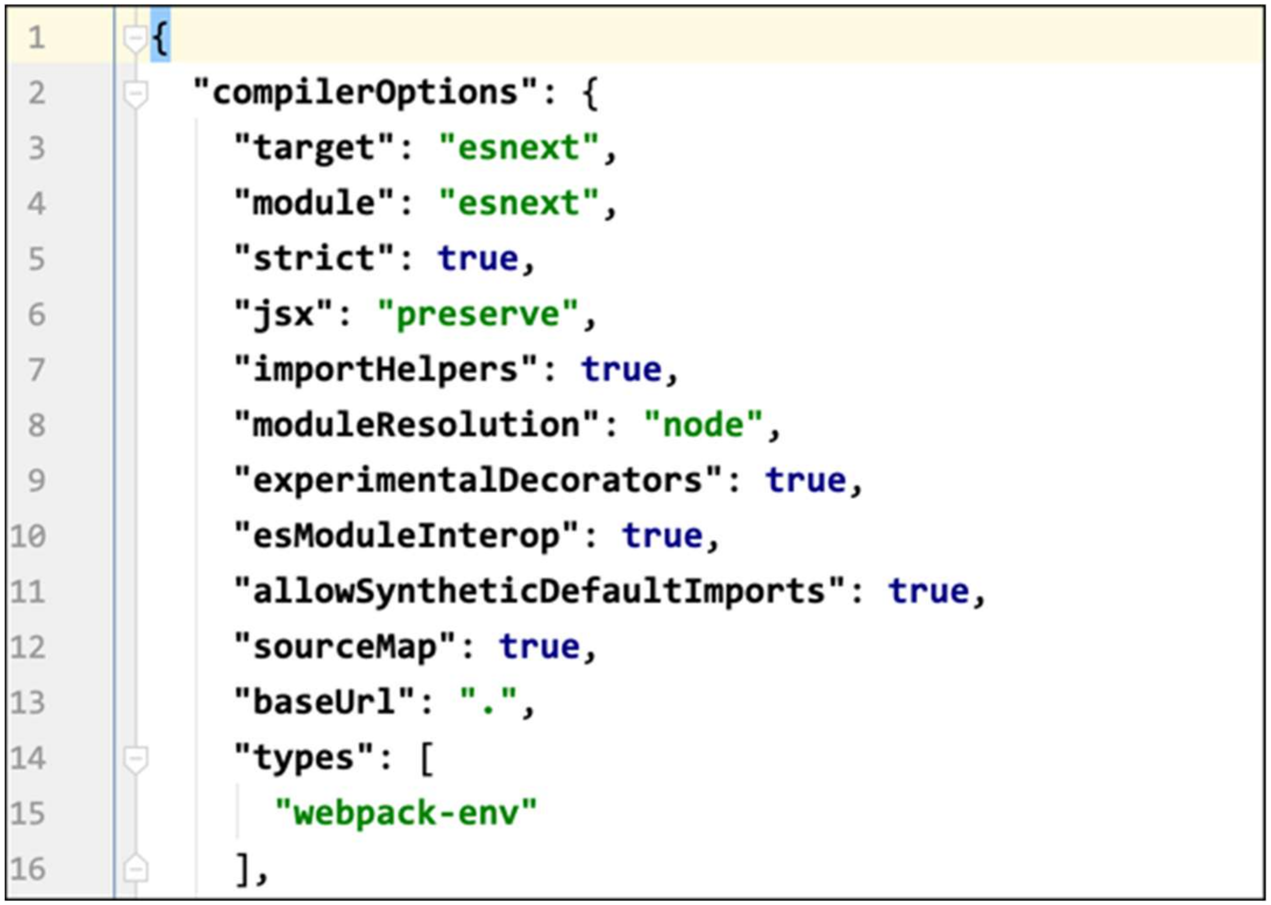
`Use class-style component syntax? > Yes`



```
? Please pick a preset: Manually select features
? Check the features needed for your project: Babel, TS, Router, Linter
? Use class-style component syntax? Yes
? Use Babel alongside TypeScript for auto-detected polyfills? Yes
? Use history mode for router? (Requires proper server setup for index fallback
in production) Yes
? Pick a linter / formatter config: TSLint
? Pick additional lint features: (Press <space> to select, <a> to toggle all, <i
> to invert selection)Lint on save
? Where do you prefer placing config for Babel, PostCSS, ESLint, etc.? In dedica
ted config files
? Save this as a preset for future projects? No


Vue CLI v3.4.1
✨  Creating project in /Users/PeterKassenaar/Desktop/vue-ts.
   Initializing git repository...
⚙  Installing CLI plugins. This might take a while...
```

Main Differences

- All `.js`-files are now `.ts`-files (`main.ts`, `router.ts`)
- Study/update `tsconfig.json`

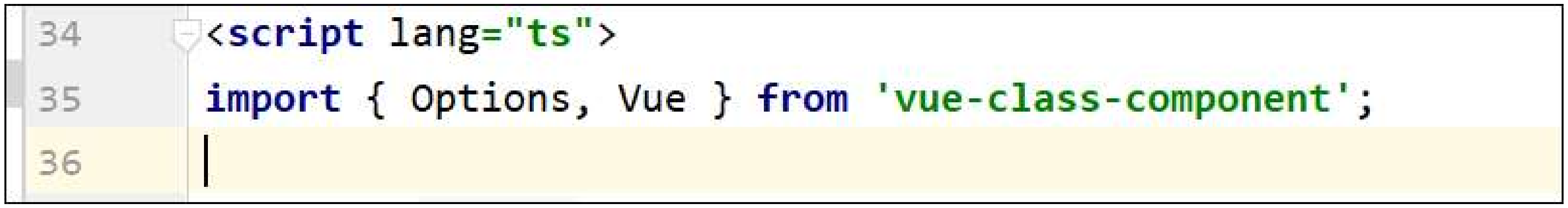


```
{
  "compilerOptions": {
    "target": "esnext",
    "module": "esnext",
    "strict": true,
    "jsx": "preserve",
    "importHelpers": true,
    "moduleResolution": "node",
    "experimentalDecorators": true,
    "esModuleInterop": true,
    "allowSyntheticDefaultImports": true,
    "sourceMap": true,
    "baseUrl": ".",
    "types": [
      "webpack-env"
    ],
```

1. When using Class style component syntax:

Components now have the `<script lang="ts">` attribute

Use `import` for decorators



```
<script lang="ts">
import { Options, Vue } from 'vue-class-component';
```

Annotate your class with `@Options()` decorator (earlier: `@Component` decorator)

Classname should be like

```
export default class <ClassName> extends Vue { …}
```

Complete class like…

```
<script lang="ts">
import { Options, Vue } from 'vue-class-component';

@Options({
  props: {
    msg: String
  }
})
export default class HelloWorld extends Vue {
  msg! : string;
}

</script>
```

2. When using `defineComponent()`

Components again have the `<script lang=“ts”>` attribute

Use `import` for `defineComponent`

```
import {CounterModel} from "@/model/counterModel";
import {defineComponent} from "vue";
```

Use the well-known `Options API` options in your component.

Component name should be like

```
export default defineComponent ({…})
```

Complete component like…

```
<script lang="ts">

import {CounterModel} from "@/model/counterModel";
import {defineComponent} from "vue";

// The Options API is used here. The component is NOT a class, but
// wrapped in defineComponent from Vue.
export default defineComponent({

  data() {
    return {
      counter: {count: 0} as CounterModel
    }
  },
  methods: {
    …
  }
});
</script>
```

OR….

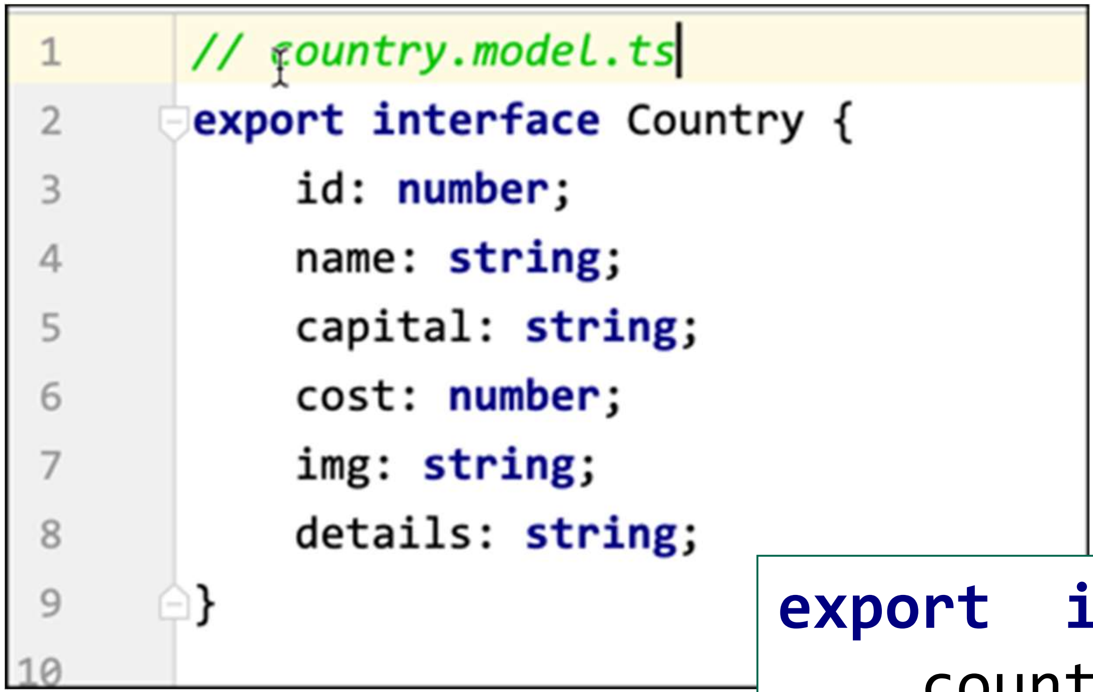
- Use `Vue.extend({…})`
- You can use the Options API as you are used to do
- In TypeScript you'll often write classes or interfaces for your data

```
<script lang="ts">
import Vue from 'vue';
import {Counter} from "@/models/counterModel";

export default Vue.extend({
  name: "Counter",
  data() {
    return {
      counter: {count: 0} as Counter
    }
  },
  ….
})
</script>
```

Importing and binding data

Create a model (`class` or `interface` or `type`)

For instance:

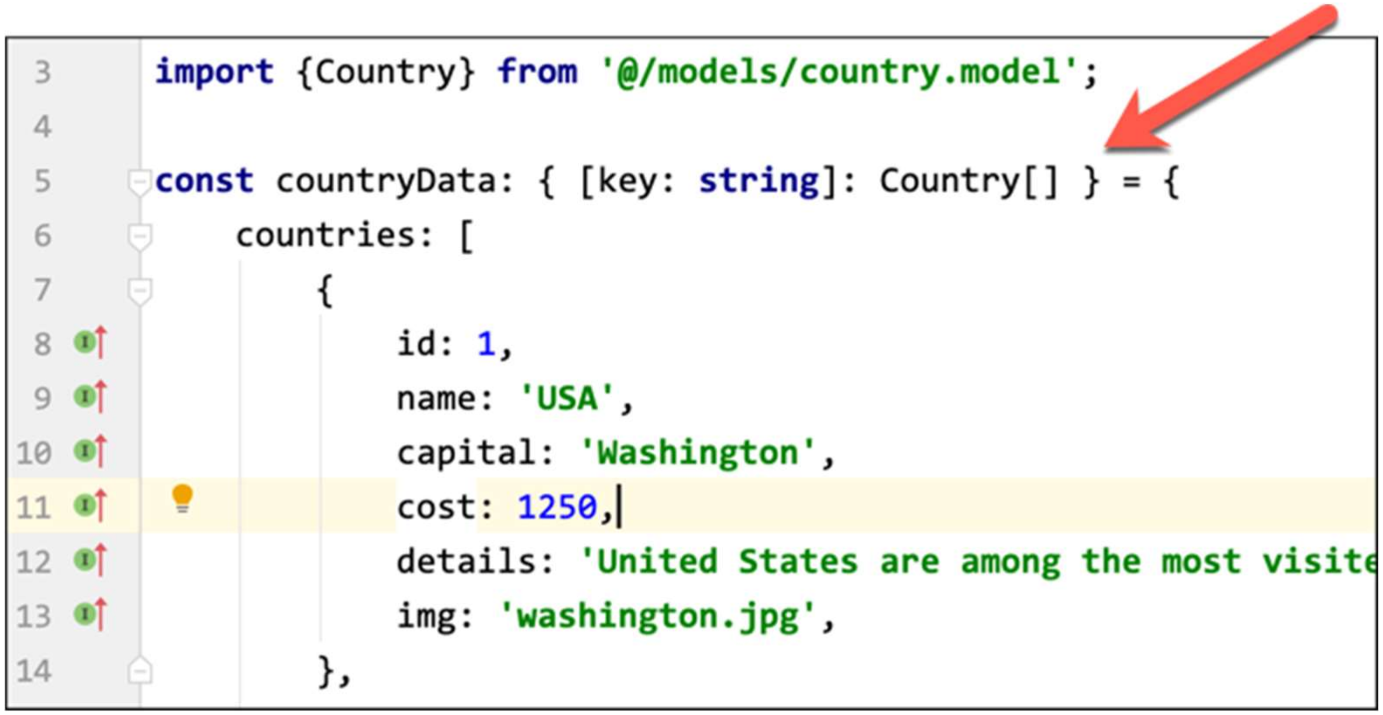


```
// country.model.ts
export interface Country {
    id: number;
    name: string;
    capital: string;
    cost: number;
    img: string;
    details: string;
}
```

```
export  interface Counter {
    count: number
}
```

Importing and binding to data

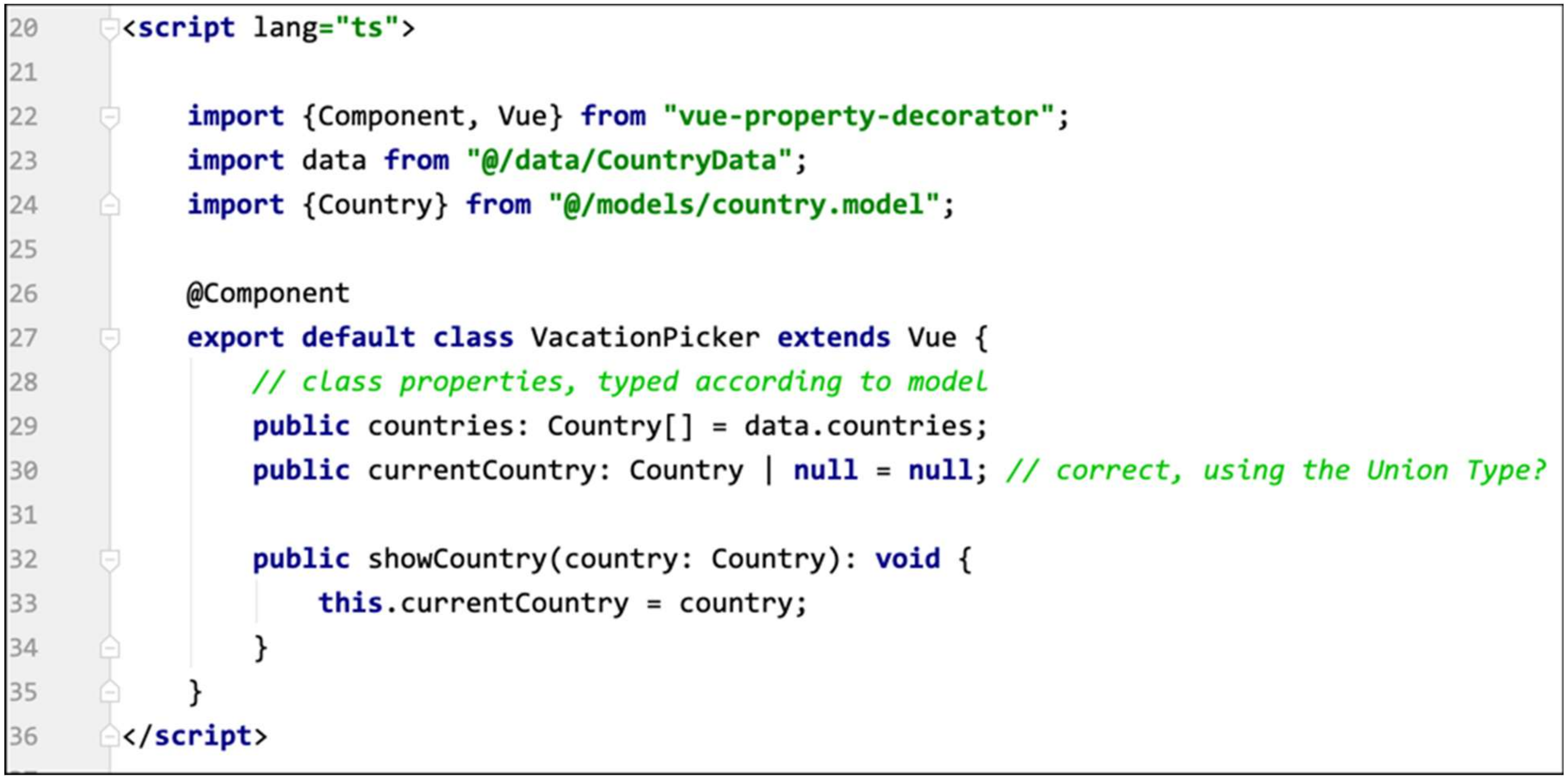
Type your data (if possible), for instance:



```
import {Country} from '@/models/country.model';

const countryData: { [key: string]: Country[] } = {
    countries: [
        {
            id: 1,
            name: 'USA',
            capital: 'Washington',
            cost: 1250,
            details: 'United States are among the most visite
            img: 'washington.jpg',
        },
```

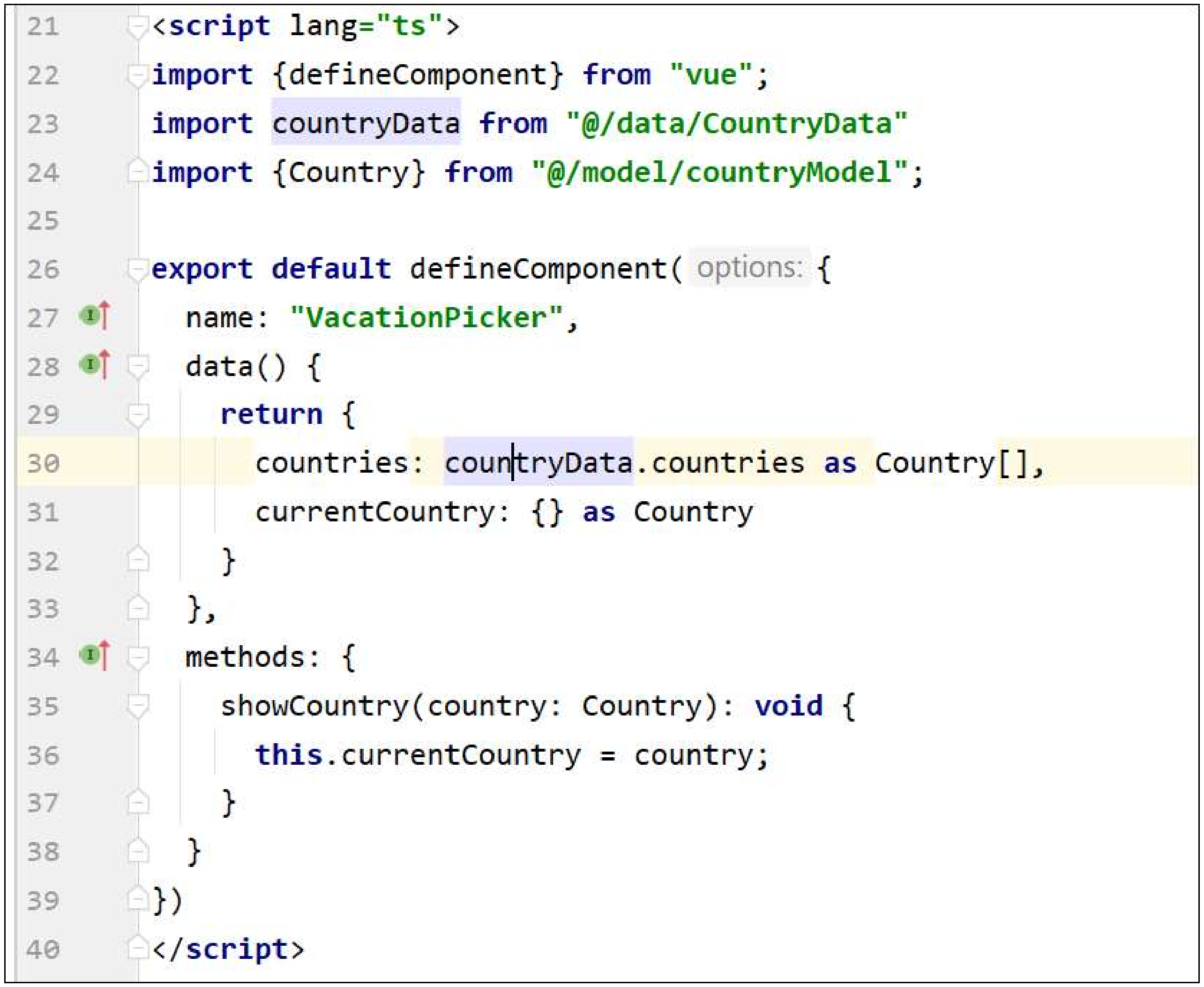
https://stackoverflow.com/questions/13315131/enforcing-the-type-of-the-indexed-members-of-a-typescript-object

Create component and import data - `class`



```
<script lang="ts">

    import {Component, Vue} from "vue-property-decorator";
    import data from "@/data/CountryData";
    import {Country} from "@/models/country.model";

    @Component
    export default class VacationPicker extends Vue {
        // class properties, typed according to model
        public countries: Country[] = data.countries;
        public currentCountry: Country | null = null; // correct, using the Union Type?

        public showCountry(country: Country): void {
            this.currentCountry = country;
        }
    }
</script>
```

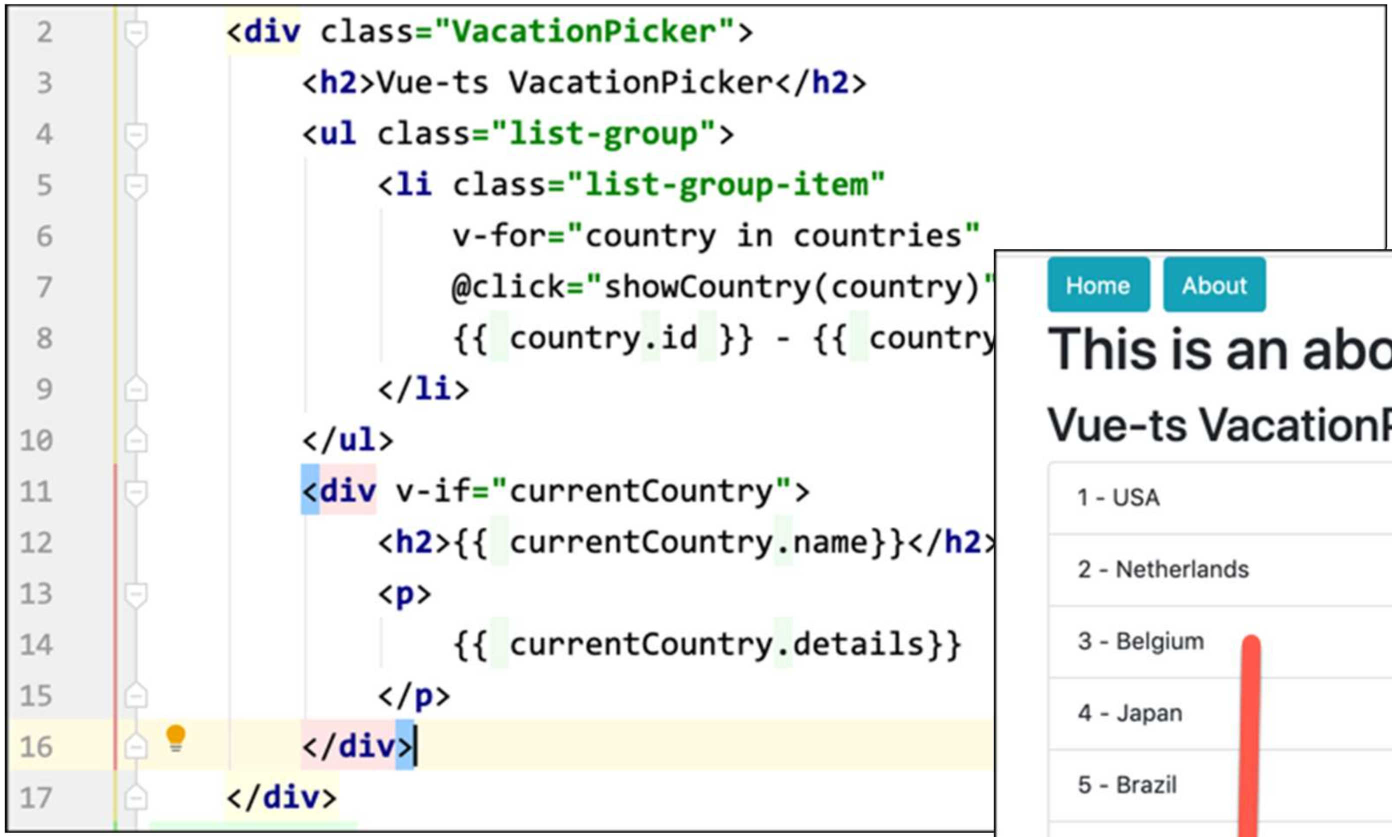
Create component and import data - defineComponent



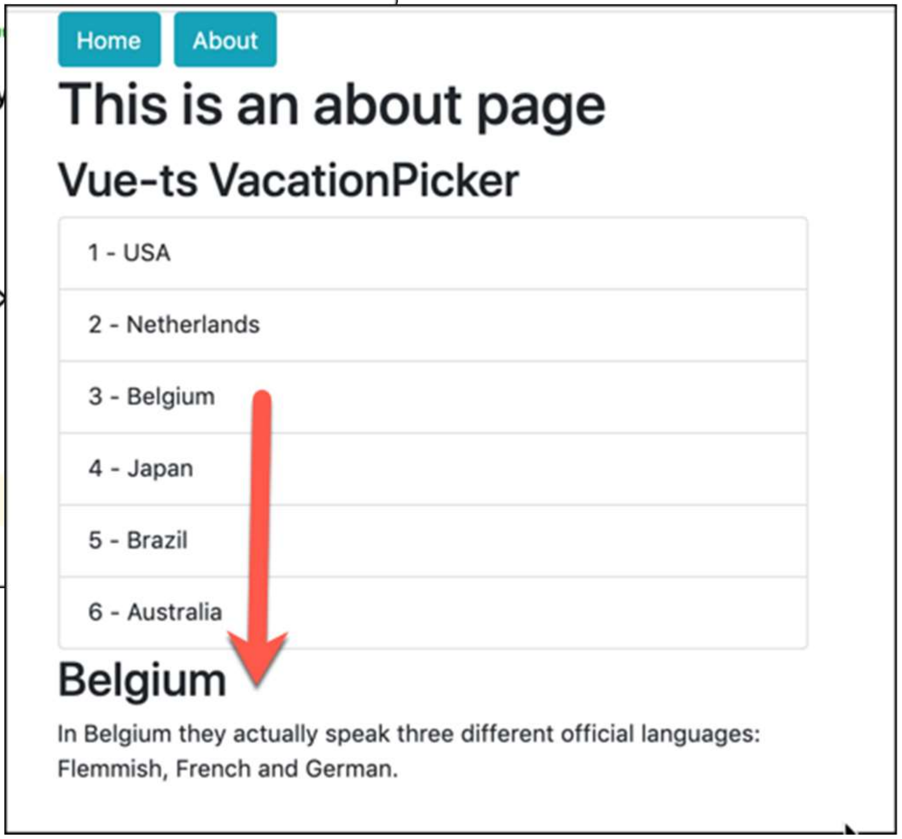
```
<script lang="ts">
import {defineComponent} from "vue";
import countryData from "@/data/CountryData"
import {Country} from "@/model/countryModel";

export default defineComponent( options: {
  name: "VacationPicker",
  data() {
    return {
      countries: countryData.countries as Country[],
      currentCountry: {} as Country
    }
  },
  methods: {
    showCountry(country: Country): void {
      this.currentCountry = country;
    }
  }
})
</script>
```

../605-vue3-ts/src/components/VacationPicker.vue

Bind to user interface as normal



```
<div class="VacationPicker">
    <h2>Vue-ts VacationPicker</h2>
    <ul class="list-group">
        <li class="list-group-item"
            v-for="country in countries"
            @click="showCountry(country)"
            {{ country.id }} - {{ country
        </li>
    </ul>
    <div v-if="currentCountry">
        <h2>{{ currentCountry.name}}</h2>
        <p>
            {{ currentCountry.details}}
        </p>
    </div>
</div>
```



This is an about page

Vue-ts VacationPicker

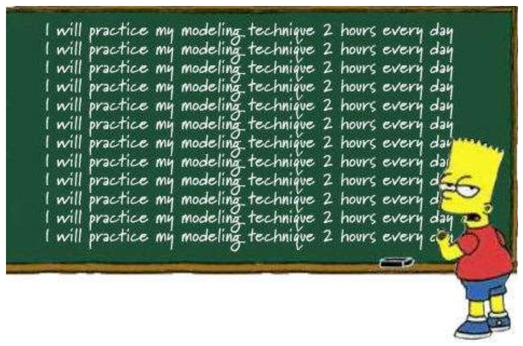
1 - USA
2 - Netherlands
3 - Belgium
4 - Japan
5 - Brazil
6 - Australia

Belgium

In Belgium they actually speak three different official languages: Flemmish, French and German.

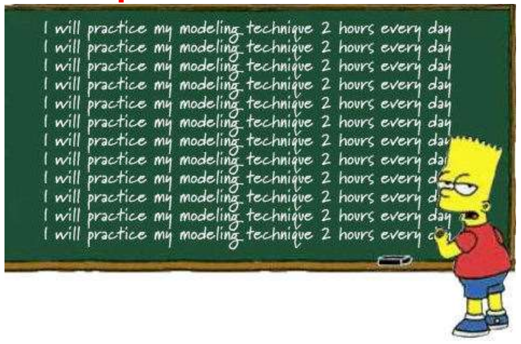
Workshop #1

- Create a new Vue+TypeScript project
- Add a component with a class
- Optional: add a component using `defineComponent()`
- Use the component(s) inside HelloWorld (or elsewhere)
- Load some data in the component, display it
- Use typesafety as much as possible
- Write an event handler and handle some event from the UI
- Example `../600-vue-ts`



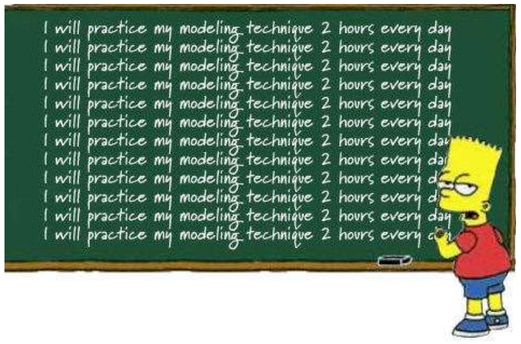
Workshop #2

- Start from `../605-vue3-ts`
- Create a prop on one of the `<Counter />` Components. Pass in the value of `Counter` as a property
- Do the same for <`VacationPicker/>`:
    - Load the list of countries in `App.vue`,
    - Pass them as a prop to `<VacationPicker />`
- Correctly type the properties on the components, possibly using `PropType`: https://v3.vuejs.org/guide/typescript-support.html#annotating-props


I will practice my modeling technique 2 hours every day
I will practice my modeling technique 2 hours every day
I will practice my modeling technique 2 hours every day
I will practice my modeling technique 2 hours every day
I will practice my modeling technique 2 hours every day
I will practice my modeling technique 2 hours every day
I will practice my modeling technique 2 hours every day
I will practice my modeling technique 2 hours every day
I will practice my modeling technique 2 hours every day
I will practice my modeling technique 2 hours every day
I will practice my modeling technique 2 hours every day
I will practice my modeling technique 2 hours every day

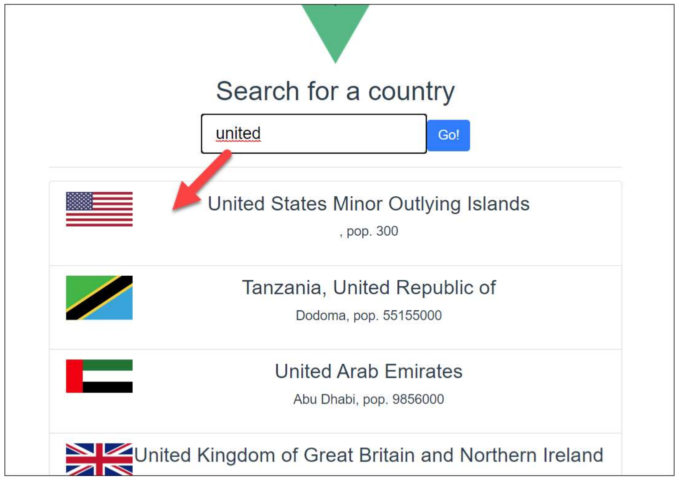
Workshop #3

- Start from `../605-vue3-ts`
- Do the same for emitting an event:
  - Emit an event from `<VacationPicker />` to `App.vue`, for instance the name of a country you click on.
  - Capture the event in `App.vue` and show which country was clicked
- Correctly type the emits, see for instance https://v3.vuejs.org/guide/typescript-support.html#annotating-emits



Workshop #4

- In your own application, create a component talking to a RESTful API (for instance https://restcountries.com/)
- Create a textbox to search for a country
- Create an interface for the returned data
- Talk to the API (using axios) to fetch countries.
- Use correct typings everywhere

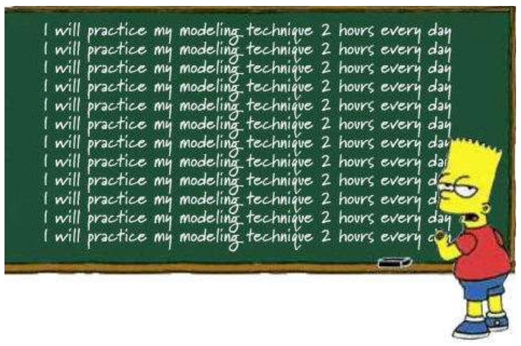


United States Minor Outlying Islands
, pop. 300

Tanzania, United Republic of
Dodoma, pop. 55155000

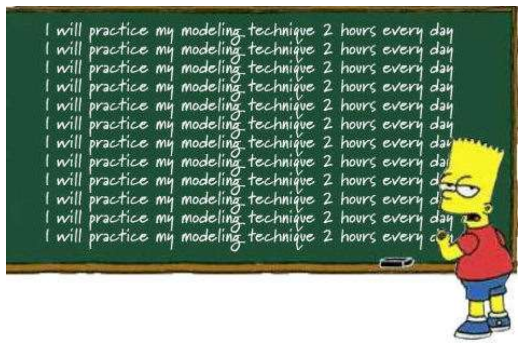
United Arab Emirates
Abu Dhabi, pop. 9856000

United Kingdom of Great Britain and Northern Ireland



Workshop #5

- Create a component talking to the dummy Users API (https://jsonplaceholder.typicode.com/users)
- Create an interface for the returned data
- Talk to the API (using axios) to fetch users.
- Show the users in the component



Checkpoint

- You know about Vue + TypeScript
- You know about different ways of declaring components, by extending classes or using `defineComponent()`
- You know how to write interfaces and type variables accordingly
- You can communicate with backends and set a type for the incoming data