jamk.fi

Web application development with Vue.js

Tomi Kumpulainen

Bachelor’s Thesis
February 2021
Information and Communication Technology
Bachelor’s Degree Programme in Information and Communications Technology

jamk.fi

Description

Author(s) Kumpulainen, Tomi	Type of publication Bachelor’s Thesis	Date February 2021
		Language of publication English
	Number of pages 56	Permission for web publication: x

Title of publication
Web application development with Vue.js

Degree programme
Bachelor’s Degree Programme in Information and Communications Technology

Supervisor(s)
Manninen, Pasi; Niemi, Kari

Assigned by
Zaibatsu Interactive Oy

Abstract

The goal of the thesis was to engineer a learning tool in the form of a web application, which could be used to educate the most important features of the Vue.js ecosystem to an employee or a trainee more efficiently than the official documentation by focusing on the SFC syntax instead of the regular syntax used in the official documentation. The thesis was assigned by Zaibatsu Interactive Oy to complement their Self Dev program, which allows the employees of the company to use 5 % of their monthly worktime for developing their professional skills in a volitional manner.

This goal was realized by carefully selecting the concepts discussed in the learning tool by critically evaluating the official documentation and designing the content of the learning tool based on the findings to build an efficient documentation covering the basics from the perspective of a SFC syntax user, which eliminates the need to interpret the official documentation on a case-by-case basis.

The learning tool was created by using the Vue.js ecosystem to build a PWA, which allows the web application to be used on any device with a standards-compliant browser and regardless of an active internet connection.

Based on the results, tangible efficiency improvements were achieved, while further development options were found.

Keywords/tags (subjects)
Vue.js, Vue CLI, Vue Loader, Vue Router, Vuex, Web application development

Miscellaneous (confidential information)

jamk.fi

Kuvailulehti

Tekijä(t) Kumpulainen, Tomi	Julkaisun laji Opinnäytetyö, AMK	Päivämäärä Helmikuu 2021
		Julkaisun kieli Englanti
	Sivumäärä 56	Verkkojulkaisulupa myönnetty: x
Työn nimi **Web-sovelluskehitys Vue.js:llä**		
Tutkinto-ohjelma Insinööri (AMK), Tieto- ja viestintätekniikka		
Työn ohjaaja(t) Manninen, Pasi; Niemi, Kari		
Toimeksiantaja(t) Zaibatsu Interactive Oy		

Tiivistelmä

Opinnäytetyön tarkoituksena oli luoda web-sovellusmuotoinen opetustyökalu, jota voitaisiin käyttää Vue.js-ekosysteemin tärkeimpien toimintojen perehdyttämiseen työntekijälle tai harjoittelijalle virallista dokumentaatiota tehokkaammin keskittymällä erityisesti SFC-syntaksiin virallisessa dokumentaatiossa käytettävän tavallisen syntaksin sijaan. Opinnäytetyön toimeksiantajana toimi Zaibatsu Interactive Oy, joka tilasi työn täydentämään yrityksen Self Dev -ohjelmaa, joka mahdollistaa yrityksen työntekijöiden käyttää 5 % kuukausittaisesta työajastaan ammattitaitonsa kehittämiseen vapaavalintaisella tavalla.

Asetettua tavoitetta lähdettiin toteuttamaan valitsemalla huolellisesti opetustyökalussa käsiteltävät aiheet virallista dokumentaatiota kriittisesti arvioimalla ja löydösten perusteellä opetustyökalun sisältöä suunnittelemalla, jonka pohjalta voitiin luoda tehokas dokumentaatio SFC-syntaksin käyttäjän näkökulmasta, mikä poistaa tarpeen virallisen dokumentaation tapauskohtaiseen tulkitsemiseen.

Opetustyökalu luotiin rakentamalla PWA-sovellus Vuen ekosysteemiä hyödyntäen, joka mahdollistaa web-sovelluksen käytön aktiivisesta internet-yhteydestä riippumatta millä tahansa laitteella, jossa on standardinmukainen web-selain.

Tuloksista voidaan päätellä, että konkreettisia tehokkuus parannuksia saavutettiin ja kohteita mahdolliselle jatkokehitykselle löydettiin.

Avainsanat (asiasanat)
Vue.js, Vue CLI, Vue Loader, Vue Router, Vuex, Web-sovelluskehitys

Muut tiedot (Salassa pidettävät liitteet)

Contents

Figures

Terminology

API

Application Programming Interface is a computing interface that defines interactions between multiple software intermediaries.

CLI

Command-Line Interface is a text-based user interface used to issue commands to execute functions of a computer program.

CPU

Central Processing Unit is the electronic circuitry within a computer that executes instructions that make up a computer program.

CSS

Cascading Style Sheets is a style sheet language that is used for describing the presentation of a document written in markup language.

DevOps

Development Operations is a set of practices that combine software development with information technology operations to optimize the production chain.

DOM

Document Object Model is an interface that treats an HTML document as a tree structure where each node is an object representing a part of the document.

DX

Developer Experience is the emotions and attitudes of a person about using a particular product, system, or a service for software development.

GUI

Graphical User Interface is a user interface that allows a person to interact with electronic devices through graphical icons instead of a text-based user interface.

HTML

Hypertext Markup Language is the standard markup language for documents designed to be displayed in a browser.

HTTP

Hypertext Transfer Protocol is an application layer protocol used by browsers and servers for data transfer in the web.

JSX

JavaScript Extensible Markup Language is an extension to the JavaScript language syntax typically used in React development.

LTS

Long-term support is a product lifecycle management policy in which a stable release of computer software is maintained longer than the standard release.

npm

Node Package Manager is a package manager for JavaScript that allows the consumption and distribution of JavaScript modules available in the remote registry.

PWA

Progressive Web Application is an application software delivered through the web and intended to work on any platform with a standards-compliant browser.

QOL
Quality of Life is the degree to which a person is healthy, comfortable, and able to participate in or enjoy life events.

Sass
Syntactically Awesome Style Sheets is a pre-processor scripting language that is compiled into CSS and uses an indent-based syntax.

SCSS
Sassy Cascading Style Sheets is the newer syntax of Sass that uses block formatting instead of the original indent-based syntax.

SEO
Search Engine Optimization is the process of improving the quality and quantity of website traffic to a website or webpage from search engines.

SFC
Single-File Component is a file with a .vue-extension that contains information about the structure, logic, and presentation of a Vue component.

SPA
Single-Page Application is a web application or website that can dynamically change the content of a webpage without the browser having to reload in between.

SRE
Site Reliability Engineering is a discipline that incorporates aspects of software engineering and applies them to the problems in infrastructure and operations.

SSR

Server-Side Rendering is a practice where the rendering of a web application or website is done by a server instead of a browser.

SVG

Scalable Vector Graphics is an XML-based vector image format for two-dimensional graphics with support for interactivity and animation.

UI

User Interface is an interface that is used for interaction and communication between a person and a computer software.

URI

Uniform Resource Identifier is an identifier used by web technologies to locate and retrieve information resources on a network or computer filesystem.

URL

Uniform Resource Locator is a type of URI that is used to reference an information resource on a network and specify its location as well as the means to retrieve it.

UX

User Experience is the emotions and attitudes of a person about using a particular product, system, or a service.

XML

Extensible Markup Language is a markup language that defines a set of rules for encoding documents in a format that is readable by both human and machine alike.

1 Introduction

The goal of the thesis was to engineer an accessible and efficient learning tool in the form of a web application, which could be used to educate the most important and select additional features of the Vue.js ecosystem to an employee or a trainee more efficiently than the official documentation by focusing specifically on SPA development with the Vue CLI and the SFC syntax instead of the regular syntax used in the official documentation to mitigate any possible opportunities for misinterpretation.

The thesis was assigned by Zaibatsu Interactive Oy to complement their Self Dev program, which allows the employees and trainees of the company to use 5 % of their monthly worktime for developing their professional skills in a volitional manner. Therefore, the learning tool had to be designed in a way that would entice a person to immerse themselves as well as allow them to grasp the basics of the Vue.js ecosystem within the timeframe outlined in the Self Dev program for maximum benefit.

The goal was realized by carefully selecting the concepts discussed in the learning tool by critically evaluating the official documentation of each individual API and designing the content and UX of the learning tool based on the findings to build a concise, consistent, and efficient documentation covering the basics from the perspective of a Vue CLI and SFC syntax user, which eliminates the need to interpret the official documentation and the importance of each concept covered by it on a case-by-case basis.

While comparing the Vue.js ecosystem to other popular JavaScript frameworks is not the primary focus of the thesis, a critical comparison to its main competitors was provided to better illustrate the benefits and drawbacks of the framework, which allows the users of the learning tool to better discern the situations in which using the Vue.js ecosystem over the competing solutions is likely to yield beneficial results.

2 Vue.js

2.1 History

Vue.js (subsequently Vue) is an open-source JavaScript framework created by Evan You, a former Google employee who at the time worked with Angular in multiple projects and wished to create a lightweight alternative to it that would retain some of its features like data binding, while simultaneously avoid the additional concepts involved when using Angular (Cromwell 2016). Vue is maintained by Evan himself and the members of the active core team (Meet the Team n.d.).

Vue is said to be a progressive framework for developing user interfaces that is incrementally adoptable, view layer focused, easy to learn and integrate with other libraries or existing projects, while also being able to power complex SPAs on its own when used together with its SFCs (Introduction n.d.a.). Vue uses a Virtual DOM that allows the real DOM to be represented as JavaScript objects that can be intelligently manipulated and updated without having to reload the webpage (Adamakis 2020).

Starting from version 0.9 titled Animatrix, which was the first initial release of Vue back in February 2014, all the following releases of Vue have been given a title that is also the name of a popular Japanese anime or manga series. Examples of this include version 1.0 (subsequently Vue 1) being titled Evangelion, version 2.0 (subsequently Vue 2) being titled Ghost in the Shell and the latest version 3.0 (subsequently Vue 3) being titled One Piece. (Cromwell 2016; v0.9.0: Animatrix 2014; v3.0.0 One Piece 2020.) All the major releases and some of the more recent minor releases of Vue have received celebration illustrations based on these titles (See Figure 1).

Figure 1. Celebration illustrations for Vue 1, Vue 2 and Vue 3 (Adapted from Vue 1 n.d.; Vue 2 n.d.; Vue 3 n.d.)

2.2 Present

At the time of writing, the latest major release of Vue is Vue 3, which was released back in September 2020 and introduced many notable new features, including a Composition API, teleport (i.e., programmatic relocation of renderable elements), and fragments (i.e., support for multiple root nodes) (v3.0.0 One Piece 2020).

While the earlier versions of Vue were written in plain JavaScript, Vue 3 is a complete rewrite of the entire Vue codebase in TypeScript, which means that while using TypeScript with Vue 3 remains optional, it now has a near perfect integration with the JavaScript superset and can be supported without any additional tooling needed (v3.0.0 One Piece 2020).

The learning tool is designed specifically for Vue 3 and the Vue CLI version 4.5.11 has been used to validate all the related information, which means that while most of the theoretical concepts discussed remain valid for Vue 2, the provided configurations and code examples cannot be used as a reference when developing with older versions. Since the learning tool itself was created by using all the technologies discussed in it, the package.json-file of the project can be used to reference the version of each individual API the disclosed information is valid for (See Appendix 1).

2.3 Comparisons

2.3.1 Adoption

Compared to its biggest competitors in the modern JavaScript framework space, Angular and React (See Figure 2), Vue is said to have great overall performance thanks to its manageable size, great documentation, the lowest initial learning curve and being the easiest to integrate with existing projects due to its incrementally adoptable feature set (Borrelli 2019).

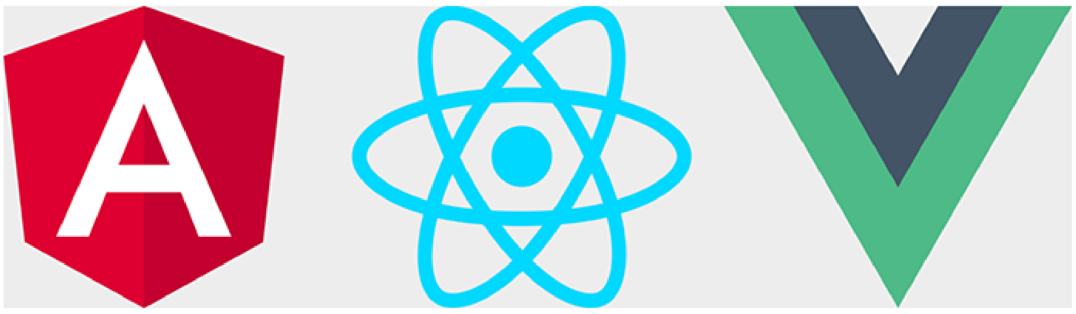


Figure 2. JavaScript frameworks Angular, React and Vue (Adapted from JavaScript frameworks n.d.)

This is by design, as Vue uses simple HTML-based templates alongside declarative rendering to express its UI, which means that any valid HTML is also a valid Vue template. React on the other hand uses JSX, an XML-like syntax extension, within render functions to achieve the same, which introduces a new syntax to the equation. The same also holds true when comparing Vue to Angular, as unlike with Angular, using TypeScript with Vue is still optional despite Vue 3 being written in TypeScript. This gives Vue a slight strategic edge over its direct competitors, as its use does not require any additional syntax adoption. (Comparison with Other Frameworks n.d.)

2.3.2 Performance

Vue is also advertised as being lightweight in terms of its size, which certainly holds true with the framework weighting in at approximately 80KB, while React and Angular both have bigger footprints at roughly 100KB and 500KB respectively (Daityari 2020). The size difference also means that Vue has the shortest average startup time and lowest memory allocation of the three, followed closely by React, while Angular can be almost twice as slow at startup and consume roughly half as much memory (Borrelli 2019).

In terms of overall performance, Vue and React can usually outperform Angular in applications where constant re-rendering of the view is needed, as they both can utilize their Virtual DOM, whereas Angular needs to modify and update the real DOM (Shah 2020). However, Angular tends to be the fastest of the three in operations that require manipulation of the real DOM, thanks to its more efficient data binding (Borrelli 2019).

2.3.3 Popularity

Trying to gauge the popularity of JavaScript frameworks can be difficult, as there does not exist a single definitive metric for determining the actual popularity of a framework but interpreting the collective results of npm package download trends, Stack Overflow developer surveys, GitHub stars and open job listings can give us a reasonably good idea of the popularity of a given framework.

With the help of npm-stat.com, a website that can generate download charts for npm packages, it can be determined that downloads between December 2018 and

December 2020 for all three selected frameworks are rising, with React leading the pack with more than 664 million downloads, followed by Vue with more than 134 million downloads and Angular placing third with almost 48 million downloads (Vorback 2018).

According to the Stack Overflow developer survey 2019, Vue and React are both the most loved and most wanted web frameworks, while Angular is the fourth least loved and third least wanted out of the twelve web frameworks listed in the survey (Stack Overflow Developer Survey Results 2019 2019).

The number of stars given to a project at GitHub can be used as a singular metric when trying to determine the popularity of a framework. This is also the first comparison where Vue can edge out both React and Angular, with Vue having over 177 000 stars, while React and Angular have roughly 161 000 and 69 000 stars, respectively. (GitHub repository for Angular 2020; GitHub repository for React 2020; GitHub repository for Vue 2 2020.)

Vue is also the newest framework of the three (Daityari 2020), meaning that it has been gaining traction among developers at a faster pace than React and Angular during its lifespan (See Figure 3), even when factoring in the GitHub stars of AngularJS, the predecessor project of Angular (GitHub repository for AngularJS 2020). However, it should be noted that the number of stars for Vue are based on Vue 2 and not Vue 3, which is the latest version that has roughly 20 000 stars at the time of writing. This is since Vue 3 is a complete rewrite of the Vue codebase in TypeScript, meaning a new repository was required to preserve the old plain JavaScript-based codebase of Vue 2. (GitHub repository for Vue 3 2020.)

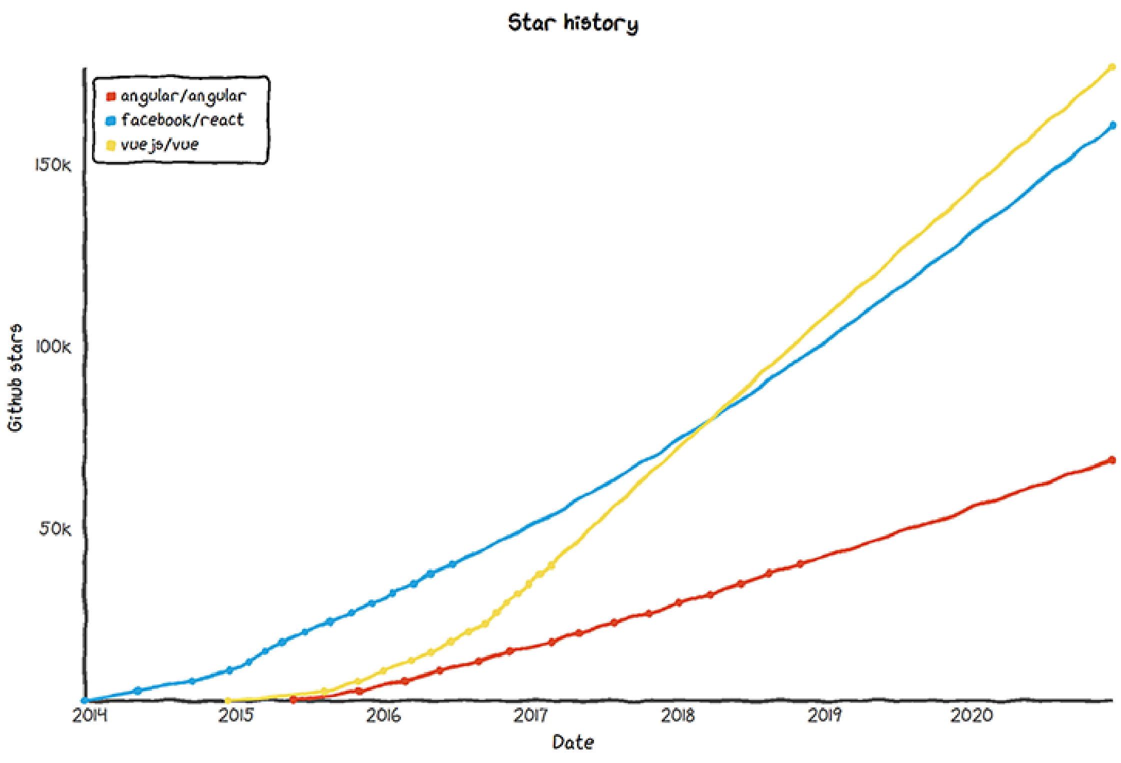


Figure 3. GitHub star history of Angular, React and Vue (Information from Star history n.d.)

When comparing open job offerings between the three, it becomes evident that despite the apparent popularity of Vue among developers, React and Angular have far more actual job demand than Vue (Neagoie 2018). The reason for this phenomenon is likely related to Vue still being a relatively new framework compared to React and Angular, which both have had a few more years to establish their status and userbase in a less competitive environment, or the fact that Vue does not have a big and well-known supporter funding its ongoing development, like Facebook and Google in the case of React and Angular, respectively.

3 Selected concepts of the learning tool

3.1 Alpha

The Alpha chapter serves as the introductory chapter of the learning tool and its main goal is to make sure that the development environment of the user meets all the requirements for Vue development. It also includes recommendations about ways to help make usage of the learning tool and Vue development in general as smooth as possible. The following concepts are covered in this chapter.

3.1.1 Node.js

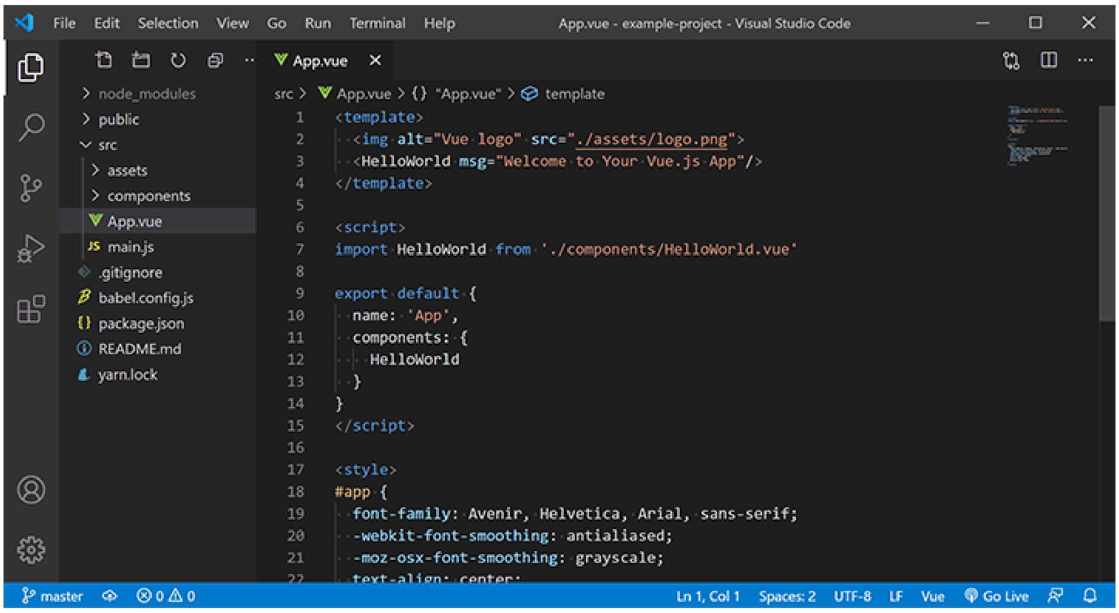
An open-source JavaScript runtime that allows code to be executed outside of a browser (GitHub repository for Node.js 2020). Node.js comes bundled with the package manager npm, which can be used to manage local dependencies of a project as well as global JavaScript tools like the Vue CLI (Ellingwood 2014). Installation of Node.js version 8.9 or above is required to use the learning tool as it is a dependency of the Vue CLI, while upgrading to the latest LTS release is recommended for the best compatibility (Installation 2020).

3.1.2 Yarn

An alternative package manager that can be used instead of npm. Yarn offers several compelling features and more concise syntax compared to npm. (Kelch 2020.) With Yarn, packages can be installed via local caching, which enables faster installation speeds and the ability to install cached packages offline (Muminovic 2019). However, using both npm and Yarn in the same project should be avoided due to resolution inconsistencies caused by unsynchronized lock files (Pile 2019). While the installation of Yarn is not required to use the learning tool or the Vue CLI, it is recommended mainly due to its more concise syntax over npm.

3.1.3 Visual Studio Code

A free source-code editor created by Microsoft for all the major operating systems, including Windows, Linux and MacOS (See Figure 4). It has support for features such as syntax highlighting, intelligent code completion, debugging and embedded Git. It also allows custom keyboard shortcuts, personalizing the UI with themes and adding functionality with extensions. (Why did we build Visual Studio Code? 2020.)



```
<template>
  <img alt="Vue logo" src="./assets/logo.png">
  <HelloWorld msg="Welcome to Your Vue.js App"/>
</template>

<script>
import HelloWorld from './components/HelloWorld.vue'

export default {
  name: 'App',
  components: {
    HelloWorld
  }
}
</script>

<style>
#app {
  font-family: Avenir, Helvetica, Arial, sans-serif;
  -webkit-font-smoothing: antialiased;
  -moz-osx-font-smoothing: grayscale;
  text-align: center;
```

Figure 4. Microsoft Visual Studio Code

According to the Stack Overflow developer survey 2019, Visual Studio Code is the most popular development environment tool not only for web development, but also for SRE and DevOps, while 0.4% of mobile developers still favor Android Studio over it (Stack Overflow Developer Survey Results 2019 2019). While the installation of Visual Studio Code is not required to use the learning tool or the Vue CLI, it is recommended as its robust features and QOL improvements can drastically improve the overall DX compared to a more conventional source-code editor.

3.1.4 Vetur

A free Visual Studio Code extension created by Pine Wu that offers support for Vue SFCs with features such as syntax highlighting, intelligent code completion, debugging and error checking (Wu 2020). While the Installation of Vetur is not required to use the learning tool or the Vue CLI, it is recommended if Visual Studio Code is used as the source-code editor since support for Vue SFCs cannot be provided without it, which has a negative impact on the overall DX.

3.2 Vue CLI

3.2.1 General

Vue CLI is the standard tooling for Vue development, and it comes with out-of-the-box support for Babel, TypeScript, PWA, Vue Router, Vuex, CSS Pre-Processors and ESLint, as well as both Unit Testing and End-to-end Testing. Vue CLI also has an optional GUI that can be used for creating and managing projects. (Overview 2019.)

3.2.2 Installation

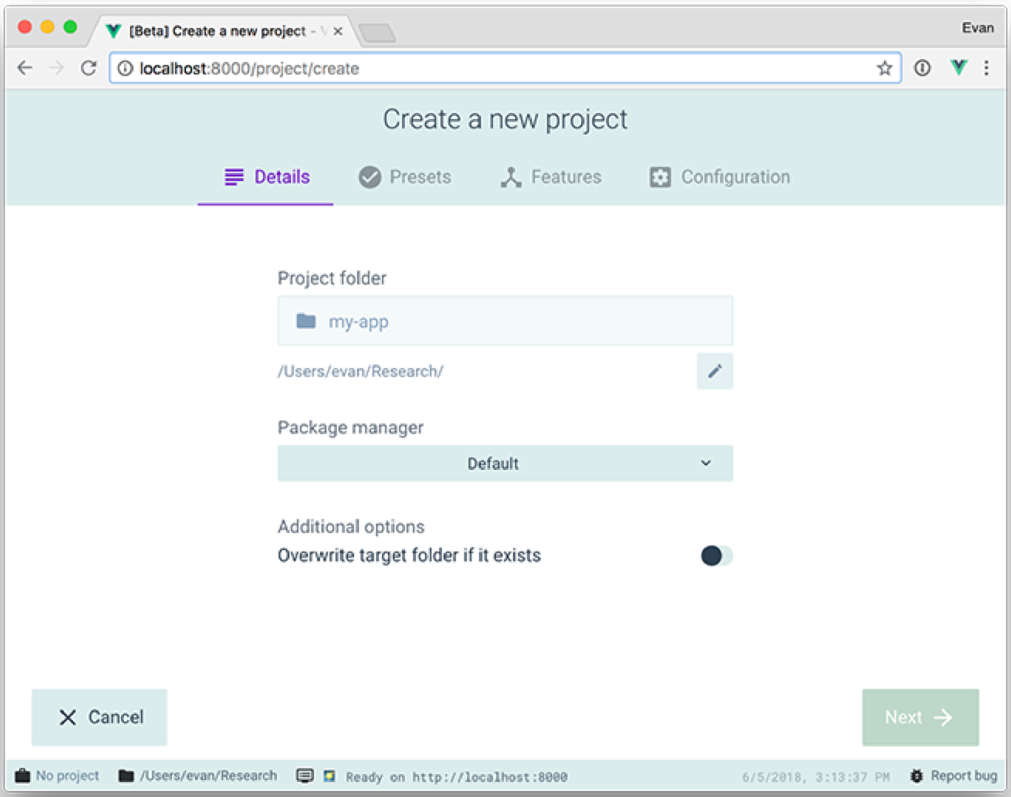
To use the Vue CLI, an installation of a global package via a package manager is required, as this will allow the Vue CLI to be invoked via the command line. Vue CLI can be installed globally with npm by using the command *npm install -g @vue/cli* or with Yarn by using the command *yarn global add @vue/cli*. (Installation 2020.)

3.2.3 Creating a project

A new Vue project can be initialized by invoking the CLI from the Node.js command line or from the built-in terminal in Visual Studio Code with the command *vue create project*, where the parameter *project* is the name of the project that is going to be created. After entering the command, the user will be prompted to pick a preset. The default preset includes Babel and ESLint, but the user can also select the additional features manually by selecting the *Manually select features* option. The latter is recommended when developing more sophisticated applications for production, while the default preset can be used for quick prototyping. The selectable features include support for Babel, TypeScript, PWA, Vue Router, Vuex, Pre-Processors, ESLint, as well as Unit and End-To-End Testing. (Creating a Project 2019.)

Manually selecting features also generates a new prompt about saving the selected features as a preset. The saved presets are stored in a JSON-file called .vuerc, which can be found from the home directory of the user, located in *%userprofile%/.vuerc* on Windows and *~/.vuerc* on Linux based systems. Other possible prompts during the project creation include a prompt about the preferred package manager that needs to be specified if the user has both npm and Yarn installed on their system, as well as a prompt to use the Taobao npm registry mirror for faster dependency installation. All saved presets can be altered by directly modifying the .vuerc-file. (Creating a Project 2019.)

Alternatively, a new Vue project can also be created by using the command *vue ui*, which will open the GUI at *localhost:8000* in the browser and then by selecting the *Create a new project here* option from the *Create* tab (Creating a Project 2019). The GUI will guide the user through the project initialization process (See Figure 5).



Project folder
my-app
/Users/evan/Research/

Package manager
Default

Additional options
Overwrite target folder if it exists

Figure 5. Creating a new project with the GUI of the Vue CLI (Adapted from GUI 2018)

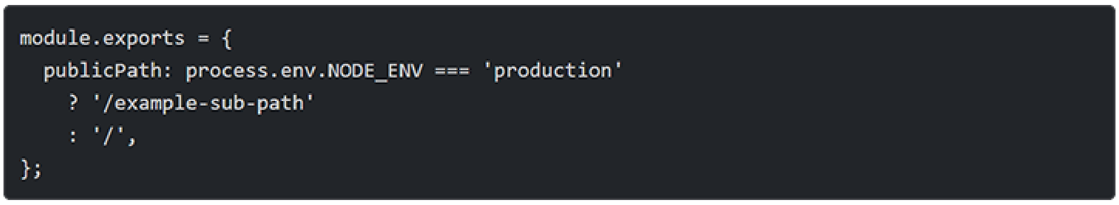
3.2.4 Building a project

The CLI Service can be used to preview a development build or to create a production build of a project. By default, using the command *npm run serve* or *yarn serve*, when using npm or Yarn respectively, will preview the project at *localhost:8080* in the browser. (CLI Service 2020.) A production build of a project can be created by using the command *npm run build* or *yarn build*, which will create a minified production-ready build of the project to the *dist* folder found at the root of the project (CLI Service 2020).

An optional *Modern Mode* can be used when creating a production build by adding the flag *--modern* at the end of the respective command. Doing so will create two separate builds of the project, with one targeting modern browsers with support for the ECMAScript 2015 specification, while the other one is a polyfilled version targeting legacy browsers. This allows the serving of a more efficient ECMAScript 2015 based build for modern browsers while still retaining the option to serve a less efficient legacy build for older browsers, instead of having to serve a single unoptimized build that accommodates for all use cases. (Browser Compatibility 2020.)

The production build created by the CLI Service is meant to be served by an HTTP server, which means that directly accessing the index.html-file of the project over the file URI scheme cannot be used to preview the production build. To preview the production build locally, a static Node.js based file server such as *serve* can be used. This can be achieved by using the command *npm install -g serve* or *yarn global add serve*, which will install *serve* as a global package and then by using the command *serve -s dist* to preview the production build. Using the optional flag *-s* deals with routing issues prevalent when trying to preview some Vue Router based builds. (Deployment 2020; GitHub repository for serve 2020.)

Depending on the production environment, creating a vue.config.js-file to the root of the project, and configuring the *publicPath* option might be required. By default, the value of *publicPath* is *'/'*, which means that the project needs to be deployed at the root of a domain (e.g., *https://www.exampledomain.com/*) and will not work if deployed at a sub-path (e.g., *https://www.exampledomain.com/example-sub-path/*). If the project needs to be deployed at a sub-path, the *publicPath* needs to be configured to reflect the sub-path. It is also possible to make *publicPath* a relative value by setting its value to *'./'*, which allows the project to be deployed under any public path. (Configuration Reference 2020.) Since the value of *publicPath* is also respected during development, it is desirable in most cases to use a conditional value to allow different values for development and production (See Figure 6).



```
module.exports = {
  publicPath: process.env.NODE_ENV === 'production'
    ? '/example-sub-path'
    : '/',
};
```

Figure 6. Using a conditional value to configure the publicPath (Information from Configuration Reference 2020)

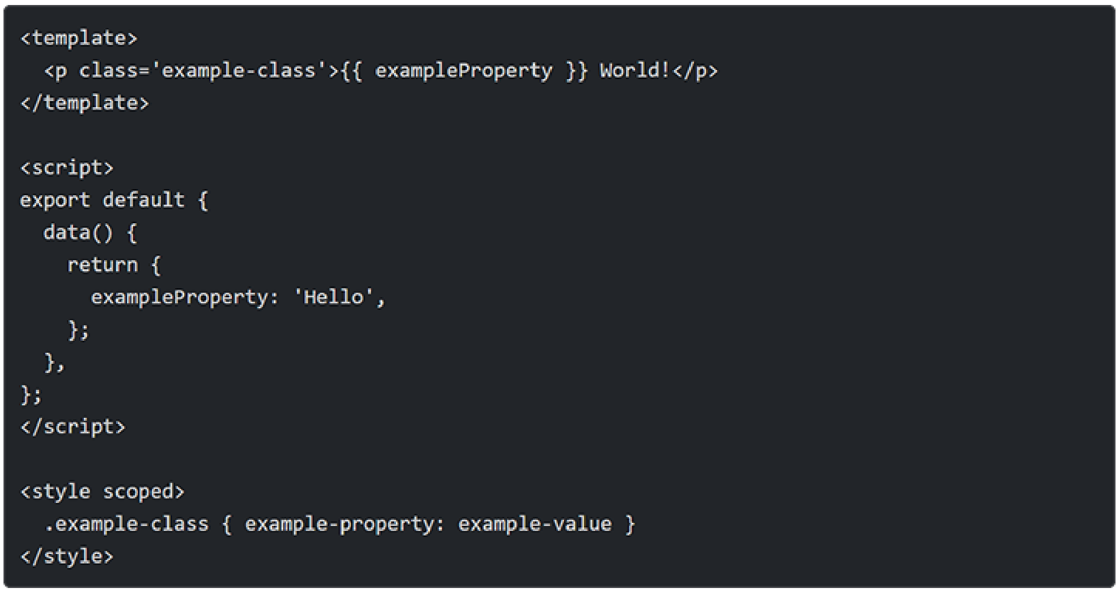
3.3 Vue Loader

3.3.1 General

Vue Loader is a webpack loader that makes the use of the SFC format and other loaders inside SFCs possible, which allows many additional features such as code transcompilation, linting and hot reloading during development (Introduction n.d.b). All projects created with the Vue CLI come with a basic pre-configuration of Vue Loader as well as all additional loaders required by the features selected during the project creation process (Getting Started n.d.a.).

3.3.2 Single-File Components

Vue SFCs are files with a .vue-extension that consist of a template block for describing the UI with HTML, a script block for adding logic with JavaScript and a style block for styling the component with CSS (See Figure 7). SFCs also have support for scoped CSS, syntax highlighting and webpack modules that allow the use of the *import* and *@import* statements within script and style blocks, respectively (Single File Components 2020). Comments can be used within all three blocks in the respective comment syntax of the used language, while top-level comments follow the conventions of HTML comments (Vue Single-File Component Spec n.d.).



```
<template>
  <p class='example-class'>{{ exampleProperty }} World!</p>
</template>

<script>
export default {
  data() {
    return {
      exampleProperty: 'Hello',
    };
  },
};
</script>

<style scoped>
  .example-class { example-property: example-value }
</style>
```

Figure 7. Hello World example with a Single-File Component (Information from Single File Components 2020)

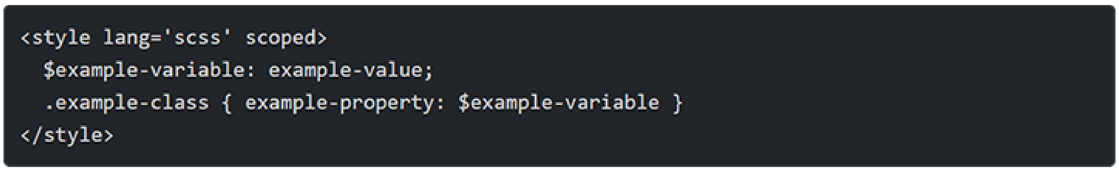
The HTML based template block allows the rendered DOM to be declaratively bound to the underlying data of the script block, which in turn allows Vue to intelligently determine the minimal number of components to re-render and DOM manipulations to apply to accommodate for a state change. Text interpolation with the “mustache” syntax, denoted by *{{ }}*, is the most basic form of said data binding and can be used to dynamically set a text within the template block to match the corresponding data property of the script block. (Template Syntax 2021.) An SFC can only contain a single template block at a time (Vue Single-File Component Spec n.d.).

The JavaScript based script block is used to enclose an options object, which holds information about the data and logic (e.g., data properties and methods) of the component. The data property of the options object is a function that returns an object, which is then called when the state changes and referenced when the view is being re-rendered. (Data Properties and Methods 2020.) An SFC can only contain a single script block at a time (Vue Single-File Component Spec n.d.).

The CSS based style block is used to style elements within the template block and can be encapsulated to the scope of the component with scoped CSS. An SFC can contain multiple style blocks at the same time with both local and global CSS declarations. (Vue Single-File Component Spec n.d.)

3.3.3 Pre-Processors

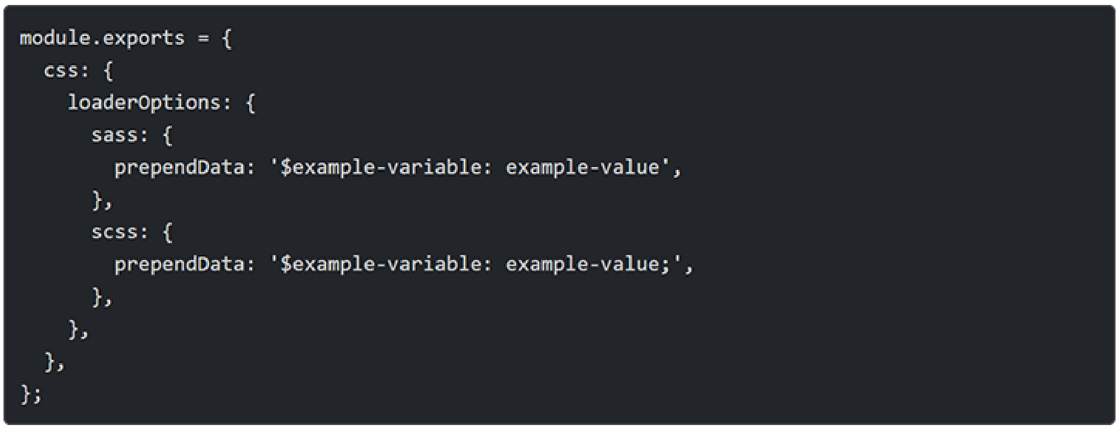
Vue Loader allows the use of CSS pre-processors like Sass within the style block of an SFC. To use a pre-processor, a corresponding loader needs to be installed via npm or Yarn and the *lang* attribute of the style block needs to be set. (See Figure 8.) If the option to use a pre-processor is selected during the interactive prompts of the project creation process, these configurations are processed automatically by the Vue CLI with no additional configuration needed (Working with CSS 2021).



```
<style lang='scss' scoped>
  $example-variable: example-value;
  .example-class { example-property: $example-variable }
</style>
```

Figure 8. Using pre-processors within a style block of an SFC (Information from Using Pre-Processors n.d.)

When using a pre-processor, it is possible to share variables and files among all SFCs via a corresponding loader option instead of having to explicitly import them in every file. This can be achieved by creating an optional vue.config.js-file to the root of the project and passing the corresponding option (e.g., *prependData* with Sass) to the loader under *css.loaderOptions*. When using Sass, it should be noted that while the loader is configured to parse both Sass and SCSS, the passed loader option still needs to follow the syntax conventions of the respective syntax. (See Figure 9.)

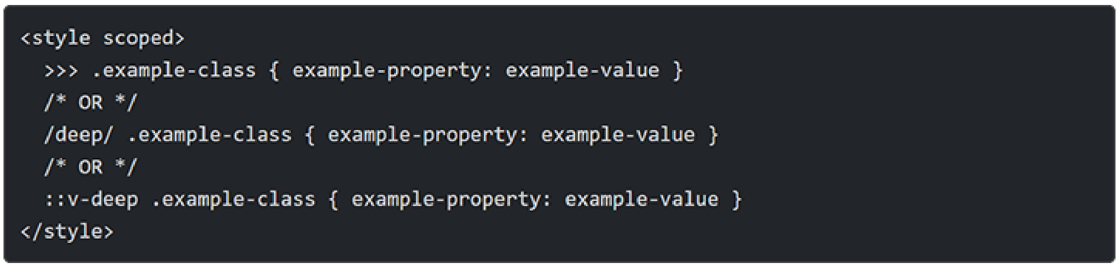


```
module.exports = {
  css: {
    loaderOptions: {
      sass: {
        prependData: '$example-variable: example-value',
      },
      scss: {
        prependData: '$example-variable: example-value;',
      },
    },
  },
};
```

Figure 9. Sharing variables by directly passing options to a loader (Information from Working with CSS 2021)

3.3.4 Scoped CSS

Vue Loader allows the use of a *scoped* attribute within a style block of an SFC, which will limit the scope of the CSS declared within the style block to that component. This prevents the style declarations of a parent component from leaking into child components. (Scoped CSS n.d.) While using separate style blocks for local and global CSS is possible, the root node of a child component can also be affected from a style block of a parent component with the *scoped* attribute by using a deep selector *>>>*, */deep/* or *::v-deep*, with the latter two being aliases of the former meant to be used with pre-processors that cannot parse the >>> selector properly (See Figure 10).



```
<style scoped>
  >>> .example-class { example-property: example-value }
  /* OR */
  /deep/ .example-class { example-property: example-value }
  /* OR */
  ::v-deep .example-class { example-property: example-value }
</style>
```

Figure 10. Using a deep selector within a scoped style block of an SFC (Information from Scoped CSS n.d.)

3.4 Directives

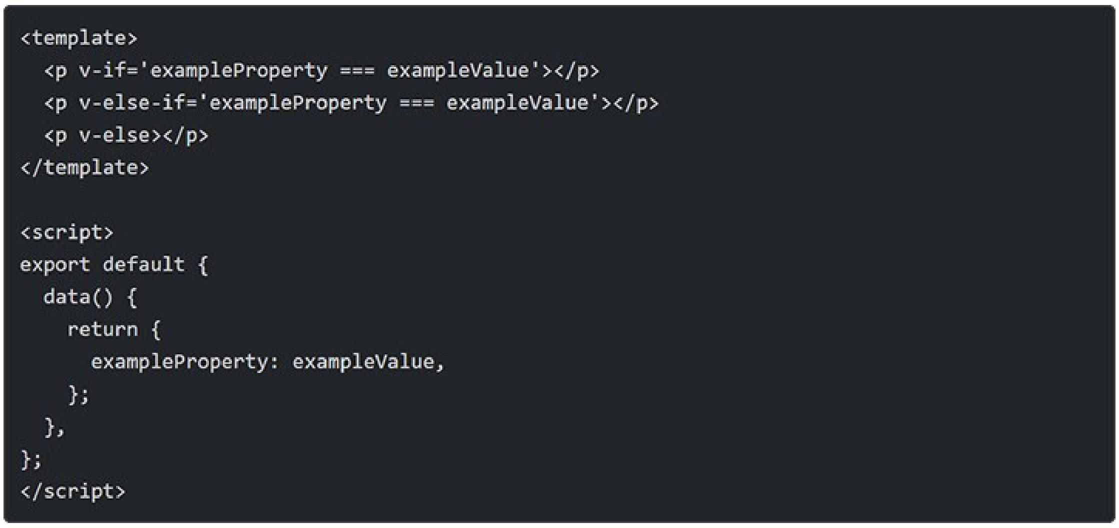
3.4.1 General

Vue uses directives to declaratively bind logic to the elements of the DOM. These directives manifest as prefixed HTML attributes, that can be broken down to the form *<element prefix-directiveID[:argument][='expression']></element>*, where the *prefix* parameter is always *v*, the *directiveID* parameter determines the type of action the directive should perform, the *argument* parameter determines e.g., the type of event or attribute the element should respond to or be bound with respectively, while the *expression* parameter refers to the data or logic (e.g., a data property or a method) used to resolve the directive during execution. The parameters enclosed in *[ ]* are either optional or situational depending on the given directive. (Directives n.d.)

3.4.2 Conditional rendering

The directives *v-if*, *v-else* and *v-else-if* can be used to conditionally render elements that only need to be rendered in certain situations. This can be achieved by passing an expression to the *v-if* directive, which causes the element to be rendered or ignored during compilation when the expression is truthful or untruthful, respectively.

(Conditional Rendering 2020.) The element with the *v-if* directive can then be immediately succeeded by an element with either the *v-else* or *v-else-if* directive or if elements with both directives are present, the element with the *v-else-if* directive must precede the element with the *v-else* directive (See Figure 11). The *v-else* directive does not expect an expression and can be used to render an alternative element when the *v-if* directive of the preceding element resolves as untruthful, whereas the *v-else-if* directive does the same but expects an expression of its own to resolve as truthful to be rendered. (Conditional Rendering 2020.)

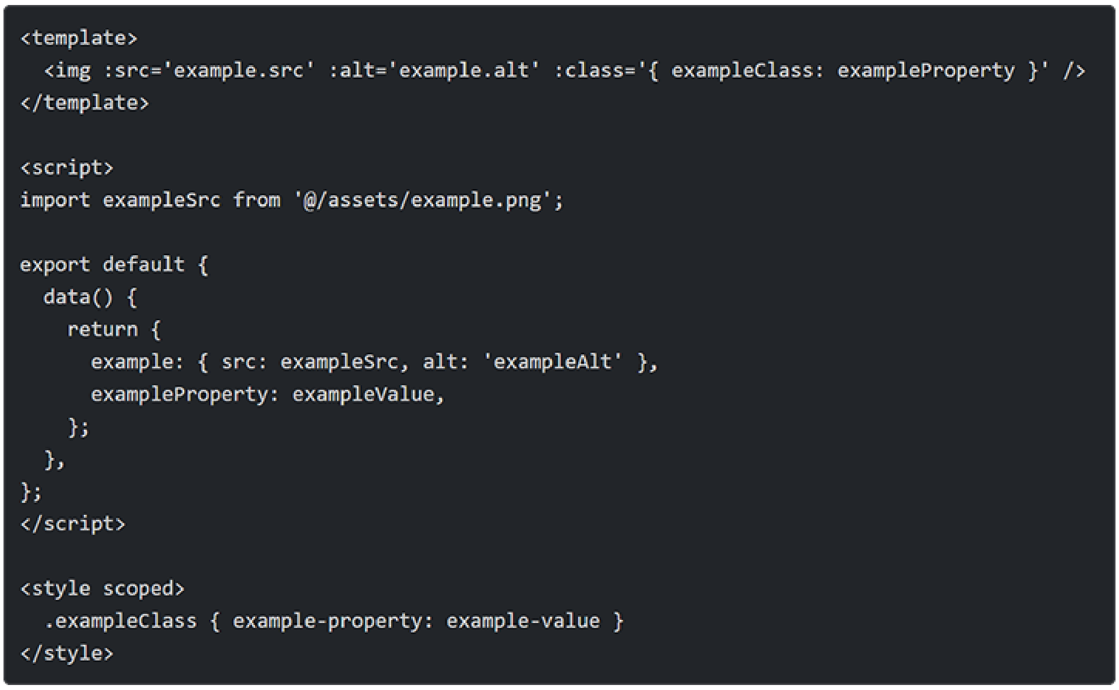


```
<template>
  <p v-if='exampleProperty === exampleValue'></p>
  <p v-else-if='exampleProperty === exampleValue'></p>
  <p v-else></p>
</template>

<script>
export default {
  data() {
    return {
      exampleProperty: exampleValue,
    };
  },
};
</script>
```

Figure 11. Conditionally rendering elements with the v-if directive (Information from Conditional Rendering 2020)

3.4.3 Attribute binding

The *v-bind* directive can be used to dynamically bind attributes to elements and props to child components. The directive expects an argument representing an attribute (e.g., *src* or *key*) or a prop and an expression representing the value of said attribute or prop. (See Figure 12.) Alternatively, an object of attributes can be passed to the directive, which allows the argument part of the directive to be omitted (Directives 2020). HTML classes can also be dynamically bound with the *class* attribute by using the argument-based syntax and passing an object containing class and data property pairs to the directive, with truthiness of the data property determining the presence of the class (Class and Style Bindings 2021). The *v-bind* directive also has a shorthand *:*, which can be used to omit the *v-bind* keyword but cannot be used when binding an object of attributes due to a syntactic incompatibility. (Directives 2020.)

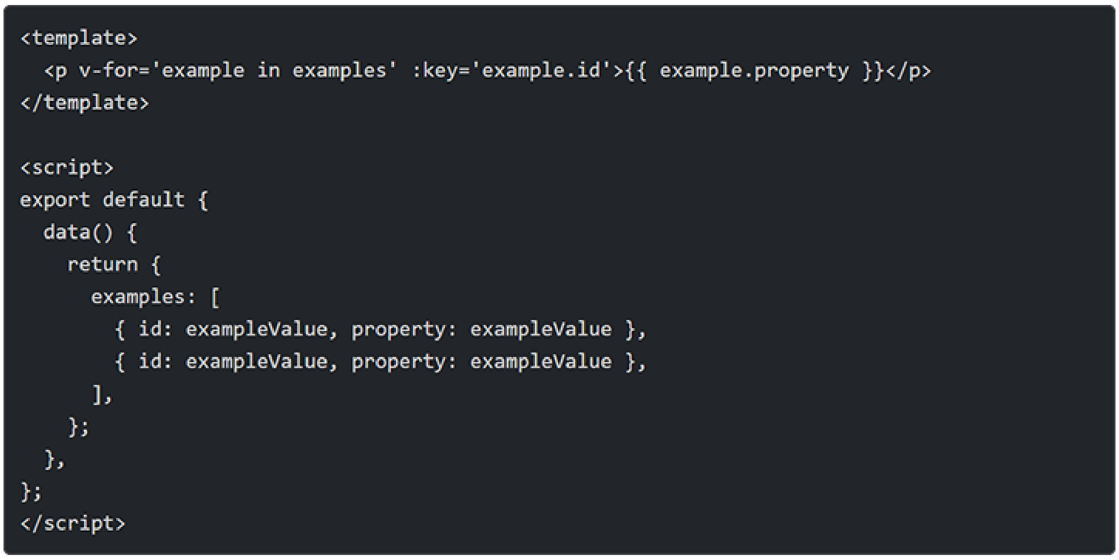


```
<template>
  <img :src='example.src' :alt='example.alt' :class='{ exampleClass: exampleProperty }' />
</template>

<script>
import exampleSrc from '@/assets/example.png';

export default {
  data() {
    return {
      example: { src: exampleSrc, alt: 'exampleAlt' },
      exampleProperty: exampleValue,
    };
  },
};
</script>

<style scoped>
  .exampleClass { example-property: example-value }
</style>
```

Figure 12. Binding attributes to elements with the v-bind directive (Information from Directives 2020)

3.4.4 Iterative rendering

The *v-for* directive can be used to iterate over the items of an array or the properties of an object and render them. The directive expects an expression in the form of *example in examples*, where the *example* parameter serves as an alias for the current iteration, the *in* parameter serves as a delimiter and the *examples* parameter is the iterable object. (See Figure 13.) An optional *index* parameter in the form of *(example, index) in examples* can be used to access the index of the iteration, while the *in* delimiter can be substituted with an *of* delimiter to match the *for…of* statement of JavaScript (List Rendering 2021). When using the *v-for* directive, a unique *key* attribute should be bound to each direct child of the element it is used on to help the Virtual DOM algorithm of Vue minimize element movement during re-rendering (List Rendering 2021; Special Attributes 2020), while using the *v-if* directive on the same element should be avoided due to its higher priority causing it being unable to access variables from the scope of the *v-for* directive (List Rendering 2021).



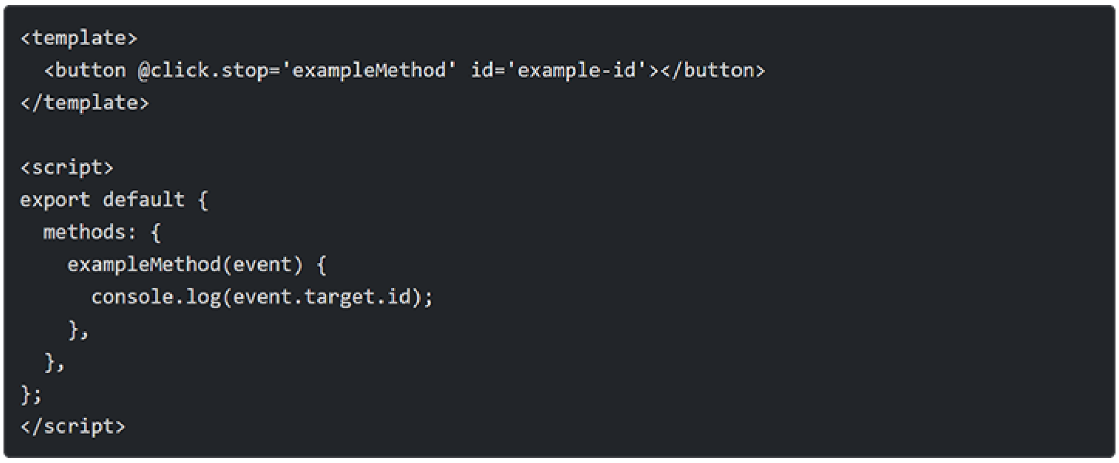
```
<template>
  <p v-for='example in examples' :key='example.id'>{{ example.property }}</p>
</template>

<script>
export default {
  data() {
    return {
      examples: [
        { id: exampleValue, property: exampleValue },
        { id: exampleValue, property: exampleValue },
      ],
    };
  },
};
</script>
```

Figure 13. Iteratively rendering the items of an array with the v-for directive (Information from List Rendering 2021)

3.4.5 Event handling

The *v-on* directive can be used to attach an event listener to an element, which allows the execution of JavaScript in response to user action. The directive expects an argument representing an event type (e.g., click or keydown) and an expression representing the logic said event should trigger (e.g., a method or an inline statement).

(See Figure 14.) The *v-on* directive supports optional event modifiers, which allow event interface methods (e.g., *preventDefault* and *stopPropagation*) to be called automatically when the event is triggered by adding a corresponding postfix (e.g., *.prevent* and *.stop*) to the directive. Key modifiers for keyboard events are also supported, which allow specific keys to be listened by converting the respective key value (e.g., Enter, Page Down) to the kebab-case format (e.g., .enter, .page-down) and adding the converted value as a postfix to the directive. The *v-on* directive also has a shorthand *@*, which can be used to omit the *v-on* keyword and the *:* preceding the argument, enabling concise syntax. (Event Handling 2020.)



```
<template>
  <button @click.stop='exampleMethod' id='example-id'></button>
</template>

<script>
export default {
  methods: {
    exampleMethod(event) {
      console.log(event.target.id);
    },
  },
};
</script>
```

Figure 14. Handling events with the v-on directive (Information from Event Handling 2020)

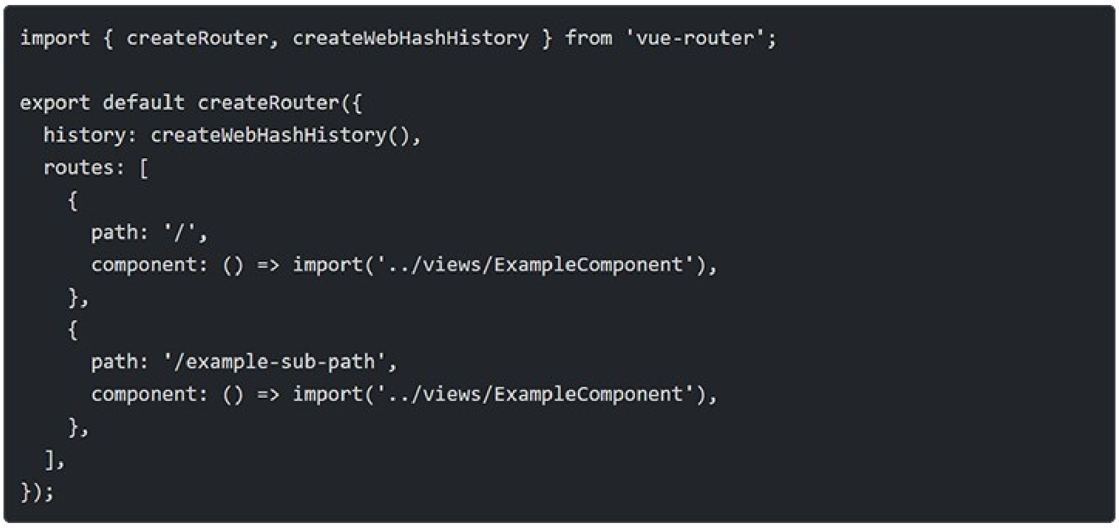
3.5 Vue Router

3.5.1 General

The official router implementation specifically designed for Vue that provides dynamic component-based routing and different history modes to accommodate for different use cases (Introduction n.d.c.). Since applications created with the Vue CLI are composed out of components, using the Vue Router to map the components to specific routes is a logical and powerful way to approach navigation in a Vue SPA (Getting Started n.d.b.).

3.5.2 Creating a router

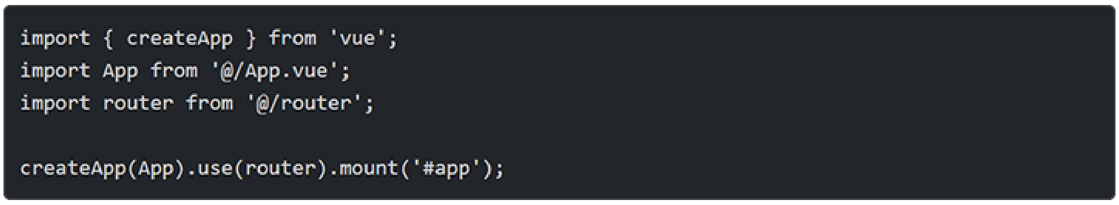
To use Vue Router, a router instance needs to be created and injected to the root component of the project, which allows the router to be accessed from all components that are children of said component (Getting Started n.d.b.). A router instance can be created by using the *createRouter* function and passing in an options object containing a *history* function and a *routes* array, which are used for specifying the history mode and the route configurations, respectively (See Figure 15). If the option to use Vue Router is selected during the interactive prompts of the project creation process, the router instance is created automatically by the Vue CLI and can be found from *src/router/index.js*.



```
import { createRouter, createWebHashHistory } from 'vue-router';

export default createRouter({
  history: createWebHashHistory(),
  routes: [
    {
      path: '/',
      component: () => import('../views/ExampleComponent'),
    },
    {
      path: '/example-sub-path',
      component: () => import('../views/ExampleComponent'),
    },
  ],
});
```

Figure 15. Creating a Vue Router instance (Information from Getting Started n.d.b.)

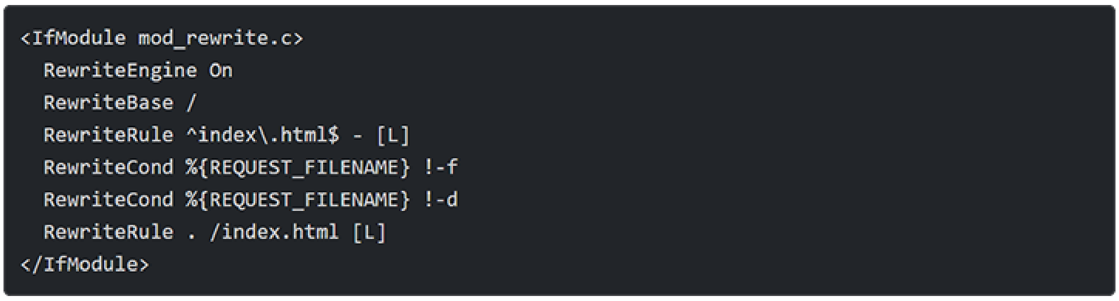
The root component of the project can be specified by using the *createApp* function and passing the component (e.g., App.vue) as an option, which can then be immediately followed by the router injection achieved by using the *use* function and passing the created router as an option (See Figure 16). If the option to use Vue Router is selected during the interactive prompts of the project creation process, the router is injected to the root component of the project automatically by the Vue CLI and the configuration can be found from *src/main.js*.



```
import { createApp } from 'vue';
import App from '@/App.vue';
import router from '@/router';

createApp(App).use(router).mount('#app');
```

Figure 16. Injecting a Vue Router instance (Information from Getting Started n.d.b.)

Vue Router supports different history modes, namely the *Hash Mode* and the *HTML Mode*, which both have their benefits and drawbacks. The *Hash Mode* that is created with the function *createWebHashHistory* uses a # character preceding the path (e.g., *https://www.exampledomain.com/#/example-sub-path*), which allows the route to be passed within the web application or website without being sent to the server. This is a workaround that has a negative impact on SEO but allows the otherwise necessary server-side configurations to be circumvented (Different History modes n.d.). The *HTML Mode* that is created with the function *createWebHistory* allows the use of a regular SEO friendly URL (e.g., *https:/www.exampledomain.com/example-sub-path*), but requires additional configuration on the server-side to prevent 404 errors when trying to access specific routes of the web application or website directly (Different History modes n.d.). This can be achieved by implementing a catch-all fallback route to the server, which will serve the index.html-file of the project when a 404 error occurs (See Figure 17). However, this is not a perfect solution as the server will no longer report 404 errors at all if a certain route does not exist. To solve this issue, a catch-all route in the form of *{ path: '/:pathMatch(.*)', component: () => import('../views/ExampleComponent') }* should be implemented to serve a dedicated 404 webpage (Different History modes n.d.).



```
<IfModule mod_rewrite.c>
  RewriteEngine On
  RewriteBase /
  RewriteRule ^index\.html$ - [L]
  RewriteCond %{REQUEST_FILENAME} !-f
  RewriteCond %{REQUEST_FILENAME} !-d
  RewriteRule . /index.html [L]
</IfModule>
```

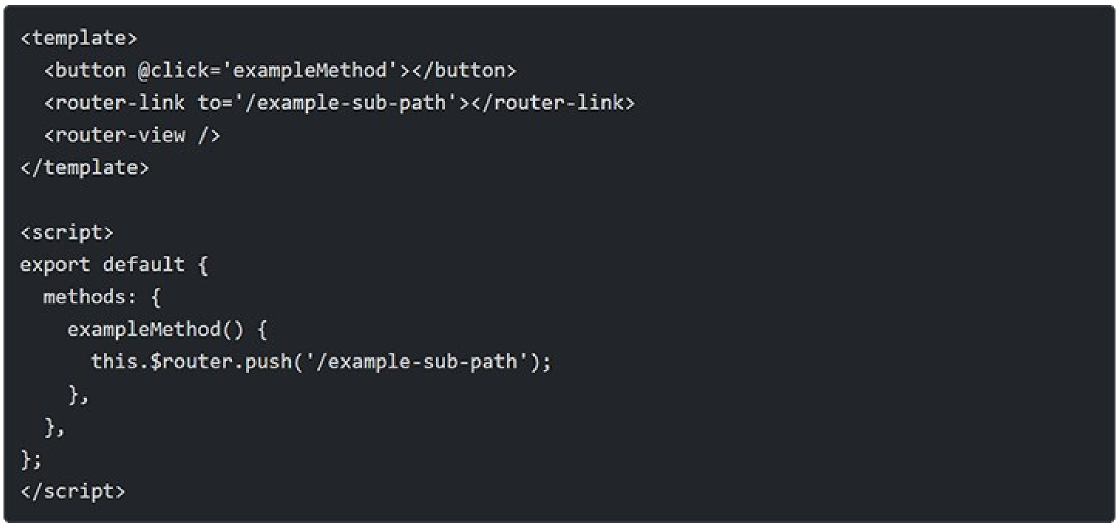
Figure 17. Apache catch-all fallback route configuration (Information from Different History modes n.d.)

Furthermore, if the project needs to be deployed at a sub-path (e.g., https://www.exampledomain.com/example-sub-path/) instead of the root of a domain, the base URL of the server configuration (e.g., *RewriteBase /* when using Apache) needs to reflect the value of the *publicPath* option of the vue.config.js-file (e.g., *RewriteBase /example-sub-path/* when using Apache) and must be passed as a parameter to the function used to create the history mode. (Different History modes n.d.; API Reference n.d.)

3.5.3 Using a router

The *router-link* custom component can be used to create links that allow the URL of the webpage to be changed without reloading the webpage by using the *to* attribute to specify the path, while the *router-view* custom component can be used to display the component that is mapped to said path (See Figure 18). The router instance can be accessed via a method with *this.$router* and then be used in conjunction with the

JavaScript *push* method to navigate to another path, while the dynamic properties of the current path can be accessed by returning a *param* via a computed property with *this.$route.params* (Getting Started n.d.b.).



```
<template>
  <button @click='exampleMethod'></button>
  <router-link to='/example-sub-path'></router-link>
  <router-view />
</template>

<script>
export default {
  methods: {
    exampleMethod() {
      this.$router.push('/example-sub-path');
    },
  },
};
</script>
```

Figure 18. Using the Vue Router to navigate to another path (Information from Getting Started n.d.b.)

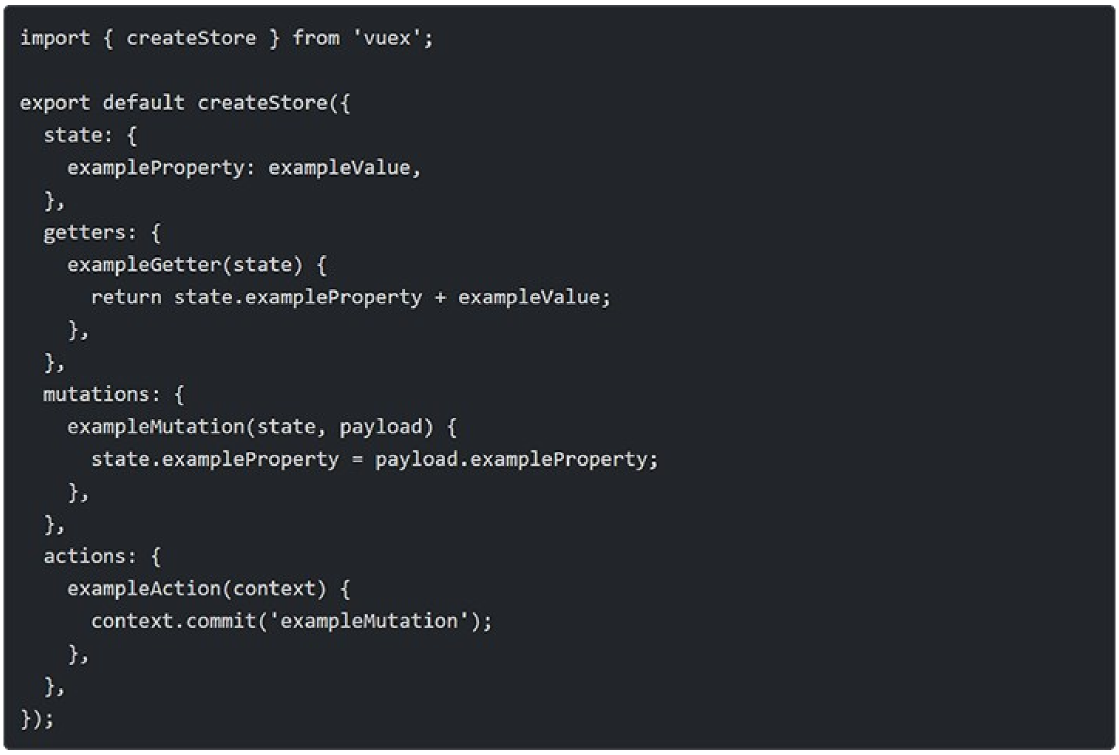
3.6 Vuex

3.6.1 General

The official Flux-like state management pattern specifically designed for Vue that provides a uniform state for all the components in an application. Vuex promotes one-way data flow that ensures predictable state mutations and enables easier communication between components, which results in more maintainable code in medium to large sized projects at the cost of more concepts to learn and boilerplate to write. (What is Vuex? 2021.)

3.6.2 Creating a store

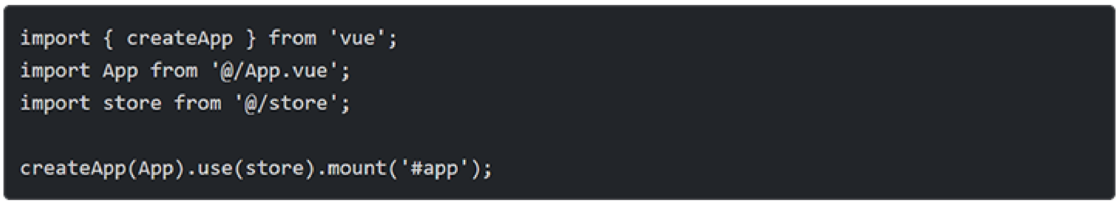
To use Vuex, a store instance needs to be created and injected to the root component of the project, which allows the store state to be accessed from all components that are children of said component. (Getting Started 2021.) A store instance can be created by using the *createStore* function and passing in an options object containing a *state* object, a *getters* object, a *mutations* object, and an *actions* object, which are used for storing the state, storing the computed state, synchronous mutation of the state and asynchronous mutation of the state, respectively (See Figure 19). If the option to use Vuex is selected during the interactive prompts of the project creation process, the store instance is created automatically by the Vue CLI and can be found from *src/store/index.js*.



```
import { createStore } from 'vuex';

export default createStore({
  state: {
    exampleProperty: exampleValue,
  },
  getters: {
    exampleGetter(state) {
      return state.exampleProperty + exampleValue;
    },
  },
  mutations: {
    exampleMutation(state, payload) {
      state.exampleProperty = payload.exampleProperty;
    },
  },
  actions: {
    exampleAction(context) {
      context.commit('exampleMutation');
    },
  },
});
```

Figure 19. Creating a Vuex store instance (Information from Getting Started 2021; Getters 2021; Mutations 2021; Actions 2021)

The root component of the project can be specified by using the *createApp* function and passing the component (e.g., App.vue) as an option, which can then be immediately followed by the store injection achieved by using the *use* function and passing the created store as an option (See Figure 20). If the option to use Vuex is selected during the interactive prompts of the project creation process, the store is injected to the root component of the project automatically by the Vue CLI and the configuration can be found from *src/main.js*.



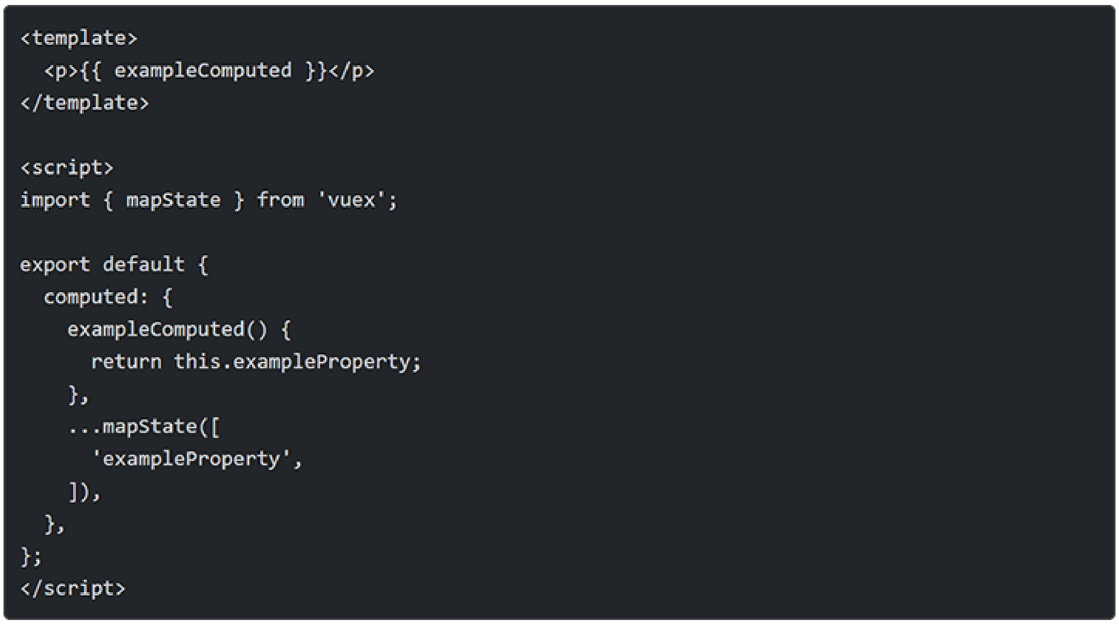
```
import { createApp } from 'vue';
import App from '@/App.vue';
import store from '@/store';

createApp(App).use(store).mount('#app');
```

Figure 20. Injecting a Vuex store instance (Information from Getting Started 2021)

3.6.3 Using a store

The store instance can be accessed from a component by returning a state or a getter property via a computed property with *this.$store.state* and *this.$store.getters* respectively, while committing a mutation or dispatching an action can be achieved via a method with *this.$store.commit* and *this.$store.dispatch* respectively (State 2021; Getters 2021; Mutations 2021; Actions 2021).

The *mapState* and *mapGetters* helper functions can be used to map store state or getter properties to the local computed properties of the component (State 2021; Getters 2021), e.g., *this.$store.state.exampleProperty* can be mapped to *this.exampleProperty* (See Figure 21), while the *mapMutations* and *mapActions* helper functions can be used to map mutations and actions to the local methods of the component, e.g., *this.$store.commit('exampleMutation')* can be mapped to *this.example-Mutation()*, which significantly reduces boilerplate when multiple store properties are needed within a single component (Mutations 2021; Actions 2021).



```
<template>
  <p>{{ exampleComputed }}</p>
</template>

<script>
import { mapState } from 'vuex';

export default {
  computed: {
    exampleComputed() {
      return this.exampleProperty;
    },
    ...mapState([
      'exampleProperty',
    ]),
  },
};
</script>
```

Figure 21. Using the Vuex mapState helper function (Information from State 2021)

3.7 Omega

The Omega chapter serves as the closing chapter of the learning tool and its main goal is to offer well informed recommendations on how the user can expand their knowledge of Vue beyond the concepts covered in the learning tool. These are concepts that are not necessary for standard Vue development but could prove useful in certain situations. The following concepts are covered in this chapter.

3.7.1 TypeScript

An open-source extension to the JavaScript language that adds static type definitions and makes it possible to catch and fix errors even before running the code. TypeScript is a superset of JavaScript that is transcompiled into JavaScript by either the TypeScript compiler or Babel. (What is TypeScript? 2020.) This means that all valid JavaScript is also valid TypeScript, which makes learning the language easier with prior experience with the former, as neglecting TypeScript syntax conventions does not prevent the transcompiled JavaScript code from being executed, but instead produces type-checking errors that can be fixed afterwards to preserve code integrity (TypeScript for the New Programmer 2020).

According to the Stack Overflow developer survey 2020, TypeScript is the second most loved and one of the most wanted programming languages of 2020 (Stack Overflow Developer Survey 2020 2020), which suggests that TypeScript has successfully established itself as one of the premiere programming languages for web application development. As mentioned in chapter 2.2, Vue 3 is a complete rewrite of the Vue codebase in TypeScript, which allows Vue to provide an enhanced support for TypeScript going forward. This means that while using TypeScript with Vue is still optional, doing so is now easier and more justifiable than ever before.

3.7.2 Server-Side Rendering

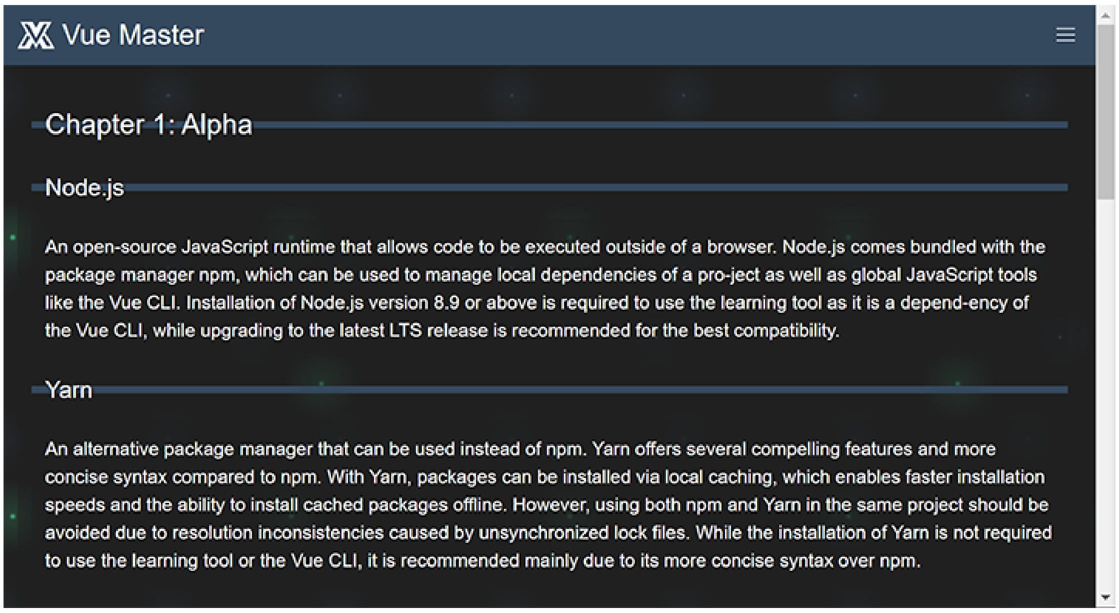
Vue is typically used to develop client-side web applications and websites made from components that manipulate the DOM in the browser, but the same can be done on the server-side by rendering the components into HTML and then sending them to the browser. When compared to a typical SPA, the benefits of SSR lie in better SEO due to search engines seeing the fully rendered webpage and faster time-to-content since a webpage rendered with SSR does not need to wait for JavaScript to be downloaded and executed. However, SSR does have some drawbacks including development constraints with certain external libraries, more complex deployment requirements and more server-side load, since rendering a whole web application or website is more CPU-intensive than serving the static files of a SPA. (Vue.js Server-Side Rendering Guide n.d.)

Furthermore, setting up a properly configured server-rendered web application or website can be significantly more complex than setting up a regular client-side SPA due to several deployment requirements. Quasar Framework (subsequently Quasar) is a higher-level framework based on the Vue ecosystem that aims to streamline the development of Vue server-rendered web applications and websites. (Server-Side Rendering n.d.). The other underlying technologies of Quasar include Capacitor, Cordova, Electron, Node.js and webpack, but having experience with these technologies is not required as they are all integrated and configured by default (Why Quasar? n.d.). This means that a person with client-side Vue experience obtained from the learning tool can start learning Quasar immediately without having to learn any other additional technologies, which makes Quasar an attractive and justifiable option going forward.

4 Creation of the learning tool

4.1 Design

The created learning tool is a web application with support for the PWA functionality, which allows the learning tool to be used on any device with a standards-compliant browser, including desktop computers, laptops, tablets, and smartphones. An active internet connection is required for the initial installation and for the first two chapters of the learning tool which include additional software and package installations, while the rest can be explored offline since the PWA functionality allows the learning tool to be cached to the used browser. The UI of the learning tool features transparent boxes over a starscape in the shades of blue and green found in official logo of Vue (See Figure 22).



Chapter 1: Alpha

Node.js

An open-source JavaScript runtime that allows code to be executed outside of a browser. Node.js comes bundled with the package manager npm, which can be used to manage local dependencies of a pro-ject as well as global JavaScript tools like the Vue CLI. Installation of Node.js version 8.9 or above is required to use the learning tool as it is a depend-ency of the Vue CLI, while upgrading to the latest LTS release is recommended for the best compatibility.

Yarn

An alternative package manager that can be used instead of npm. Yarn offers several compelling features and more concise syntax compared to npm. With Yarn, packages can be installed via local caching, which enables faster installation speeds and the ability to install cached packages offline. However, using both npm and Yarn in the same project should be avoided due to resolution inconsistencies caused by unsynchronized lock files. While the installation of Yarn is not required to use the learning tool or the Vue CLI, it is recommended mainly due to its more concise syntax over npm.

Figure 22. Chapter 1 of the learning tool viewed in a browser

To properly support the PWA functionality on a multitude of devices, a set of icons in specific sizes and formats was designed. These icons portray the name of the learning tool, *Vue Master*, in a formation that creates a layered silhouette of the letters *V* and *M* to form an icon that is reminiscent to the original logo of Vue. The set includes icons in sizes 512x512, 192x192, 180x180, 150x150, 32x32, 16x16 and an all-black version of the largest icon in the SVG format. The 512x512 and 192x192 icons are used by Google Chrome via a manifest.json-file, the 180x180 icon has a background is used by some iOS-based devices, the 150x150 icon is used by some Windows-based devices via a browserconfig.xml-file, the 32x32 icon is used by Safari and the 16x16 icon is used by older browsers which cannot use the other higher quality icons, while the SVG icon is used for the pinned tab functionality of Safari. (See Figure 23.)

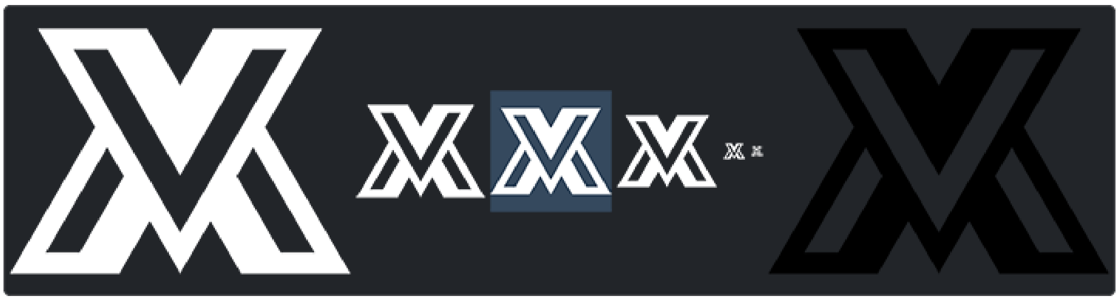
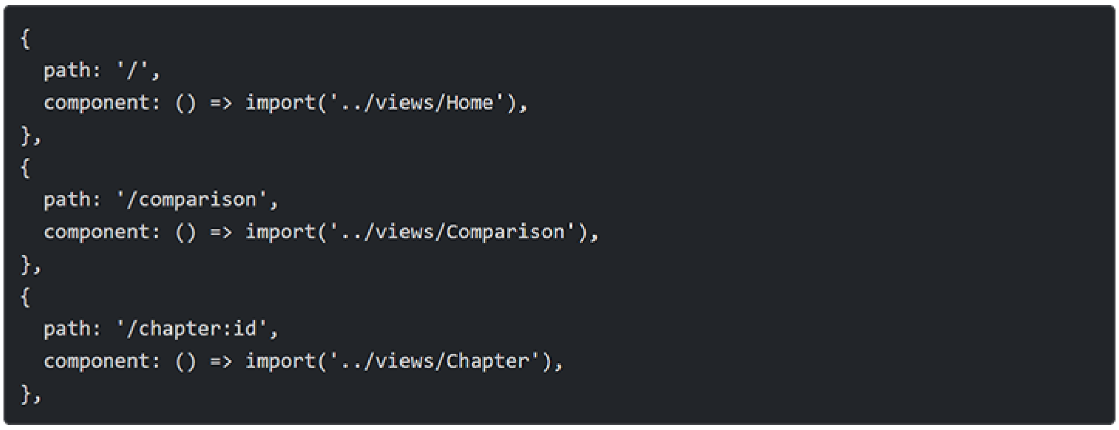


Figure 23. The set of icons designed for the learning tool

4.2 Implementation

The learning tool was created by using the Vue CLI to initialize a Vue 3 based project with support for Babel, PWA, Vue Router, Vuex, Pre-processors and ESLint. The Flexbox-based CSS Framework Bulma was also added to the project to help realize a responsive UI for the learning tool, while Sass was used as the pre-processor to complement Bulma and its modular capabilities. Yarn was selected as the package manager for the project due to its more concise syntax over npm, while the Airbnb configuration of ESLint was used to ensure code integrity.

The route configuration of the learning tool consists of three main routes, with one of them having a path with the dynamic property *id*, which is used to dynamically render all the individual chapters with just a single route and a single component. This is achieved by utilizing links that will pass the number of the target chapter as a parameter (e.g., 1, 2, 3) to a method, which will construct the path of the route by utilizing the given parameter and then navigate to the route by using *this.$router* in conjunction with the JavaScript *push()* method. (See Figure 24.)



```
{
  path: '/',
  component: () => import('../views/Home'),
},
{
  path: '/comparison',
  component: () => import('../views/Comparison'),
},
{
  path: '/chapter:id',
  component: () => import('../views/Chapter'),
},
```

Figure 24. The route configuration of the learning tool

The route will render a *Chapter* component, which will access the value of the *id* property of the path by returning it as a *param* via a computed property with *this.$route.params.id*. This allows the component to utilize the *id* property as an index to retrieve the resources of the corresponding chapter via a multidimensional array containing the resources as objects of key and value pairs, where the key represents the type of the resource (e.g., text, image) and the value is the actual resource (e.g., a paragraph of text, an image file). The retrieved resources are then iteratively rendered by using the *v-for* directive in conjunction with the *v-if* family of directives to distribute the resources to corresponding elements (e.g., *p* element for a text, *img* element for an image).

4.3 Testing

To test the performance, accessibility, utilization of best practices, SEO, and support for the PWA functionality of the created learning tool, the web application was served via the Student-server of JAMK and then audited for both mobile and desktop by using the Lighthouse audition tool (See Figure 25). According to the results of the audition, the performance of the learning tool scored 81 and 98 points on mobile and desktop respectively, with the most significant improvements being attainable by adding preload key requests, which allow the optimization of the resource request chain. The SEO of the learning tool scored 92 points on both mobile and desktop, which is a great result considering that the learning tool currently relies on the *Hash Mode* of the Vue Router, which has a negative impact on SEO. The accessibility, utilization of best practices and support for the PWA functionality of the learning tool were all excellent, with the former two scoring a perfect 100 on both mobile and desktop, while the support for the PWA functionality scored three out of three in fastness and reliability, three out of three in installability and seven out of eight in PWA optimization.

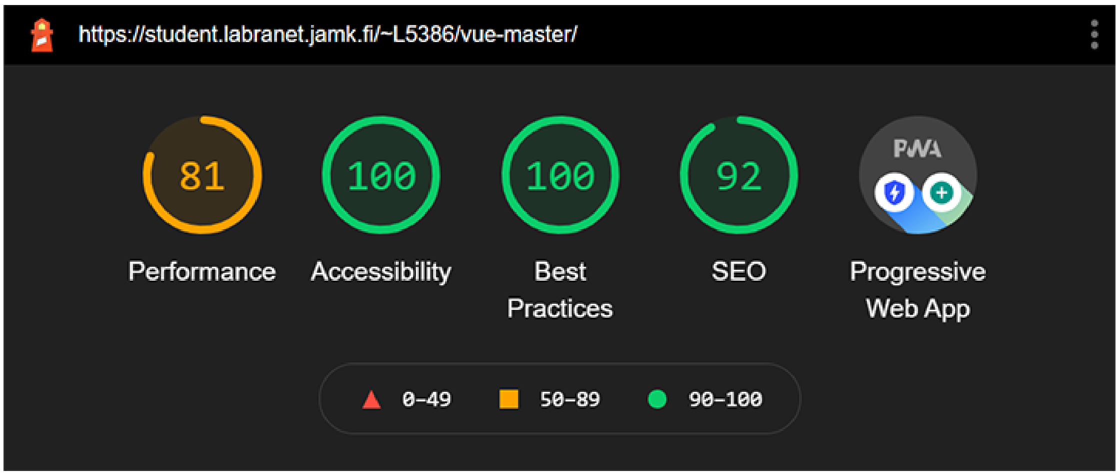


Figure 25. Lighthouse audition results of the learning tool in the mobile test

5 Results

As mentioned in chapter 1, the goal of the thesis was to engineer an accessible and efficient learning tool in the form of a web application, which could be used to educate the most important and select additional features of the Vue ecosystem to an employee or a trainee more efficiently than the official documentation by focusing specifically on SPA development with the Vue CLI and the SFC syntax instead of the regular syntax used in the official documentation to mitigate any possible opportunities for misinterpretation.

This goal was fulfilled by creating a PWA and by carefully selecting the concepts discussed in the learning tool by critically evaluating the official documentation of each individual API and designing the content and UX of the learning tool based on the findings to build a concise, consistent, and efficient documentation covering the basics from the perspective of a Vue CLI and SFC syntax user, which eliminated the need to interpret the official documentation and the importance of each concept covered by it on a case-by-case basis. However, to provide tangible evidence of the claimed efficiency improvements, direct comparisons between the official documentation of each individual API and the learning tool must be made. The following observations exhibit the various ways these improvements have been achieved.

The learning tool focuses on web application development with the Vue CLI, which allows sophisticated SPAs to be built with the help of Vue SFCs. The Vue SFCs use a syntax that differs from the regular Vue syntax used in the official documentation, which means that users of the Vue CLI must interpret the official documentation and adapt their code in a corresponding manner. This leaves room for user error and has a negative impact on DX if the user is unable to interpret the official documentation correctly. The learning tool solves this problem by presenting all concepts from the perspective of a Vue CLI user and by using the SFC syntax for all code examples.

Furthermore, since the official documentation of each individual API in the ecosystem is a collaborative effort by different contributors, the used vocabulary and the presentation of the code examples suffer from consistency issues. This problem is solved in the learning tool by using consistent vocabulary across all discussed concepts and by providing meticulously designed and uniform code examples that follow the rulesets outlined in the official Vue style guide and by the Airbnb configuration of ESLint. When creating the code examples, minimal commenting was applied to allow the code examples to be purely about logic, while the theoretical basis needed to understand the code examples is provided in the preceding paragraphs.

Another existing problem in the official documentation is the introduction of concepts that require knowledge of another concept to be executed in the recommended way, e.g., in the directive section of the documentation, the *v-for* directive is introduced before the *v-bind* directive. This is counterproductive, as the proper use of the *v-for* directive requires a *key* attribute to be bound to the direct child elements of the element with the *v-for* directive. The learning tool solves this problem by introducing concepts in a logical order to avoid any confusion caused by such situations.

When configuring the Vue Router and Vuex, it became inherently apparent that the configurations outlined in official documentations for both APIs and the actual configurations applied by the Vue CLI do not align with one other. This is since the Vue CLI resorts to a more modular approach and splits the configurations over multiple files, which results in more concise syntax when both APIs are used due to the overlap in the configuration when injecting the respective router or store instance to the root component of the project. The learning tool provides a consistent configuration style for both APIs that avoids the unnecessary confusion caused by the inconsistent documentation and manages to provide a solution that is more efficient than either of the configurations outlined in their respective documentations.

6 Conclusion

Based on the results outlined in chapter 5, the original goals of the thesis set by the client Zaibatsu Interactive Oy were achieved by creating an accessible and efficient learning tool in the form of a PWA, which provides all the basics needed for web application development with the Vue ecosystem by using the Vue CLI and the SFC syntax as the basis of the learning tool. Additional benefits were also achieved by providing a critical comparison to the direct competitors of Vue, support for setting up the development environment, recommendations about useful development tools and suggestions about learning other worthwhile technologies that complement Vue.

While the engineering of a web application can be thought as a relatively easy task on its own, attempting to optimize the configuration and use of an entire technological ecosystem was an ambitious and daunting undertaking. However, the process proved to be extremely valuable from the perspective of developing personal professional skills, since optimizing and documenting the individual aspects of the Vue ecosystem required extensive familiarization of the concepts and mastery of the special terminology. While keeping the information of the learning tool up to date will require additional resources in the future, doing so provides an easy way to familiarize a new employee or trainee with the most important features of the Vue ecosystem.

References

Actions. 2021. Official Vuex documentation about actions. Accessed on 31 January 2021. Retrieved from https://next.vuex.vuejs.org/guide/actions.html.

Adamakis, F. 2020. Vue Virtual Dom. An article about the Vue Virtual DOM. Accessed on 23 Decemeber 2020. Retrieved from https://medium.com/js-dojo/vue-virtual-dom-13af62d2be41.

API Reference. N.d. Official Vue Router API Reference. Accessed on 6 February 2021. Retrieved from https://next.router.vuejs.org/api/.

Borrelli, P. 2019. Angular vs. React vs. Vue: A performance comparison. A blogpost comparing JavaScript frameworks. Accessed on 22 December 2020. Retrieved from https://blog.logrocket.com/angular-vs-react-vs-vue-a-performance-comparison/.

Browser Compatibility. 2020. Official Vue CLI documentation about browser compatibility. Accessed on 14 January 2021. Retrieved from https://cli.vuejs.org/guide/browser-compatibility.html#modern-mode.

Class and Style Bindings. 2021. Official Vue documentation about class and style bindings. Accessed on 28 January 2021. Retrieved from https://v3.vuejs.org/guide/class-and-style.html.

CLI Service. 2020. Official Vue CLI documentation about the CLI Service. Accessed on 9 January 2021. Retrieved from https://cli.vuejs.org/guide/cli-service.html.

Comparison with Other Frameworks. N.d. Official comparison by the Vue community to competing frameworks. Accessed on 23 December 2020. Retrieved from https://vuejs.org/v2/guide/comparison.html.

Conditional Rendering. 2020. Official Vue documentation about conditional rendering. Accessed on 24 January 2021. Retrieved from https://v3.vuejs.org/guide/conditional.html.

Configuration Reference. 2020. Official Vue CLI documentation about configuration. Accessed on 15 January 2021. Retrieved from https://cli.vuejs.org/config/.

Creating a Project. 2019. Official Vue CLI documentation about creating a project. Accessed on 4 January 2021. Retrieved from https://cli.vuejs.org/guide/creating-a-project.html#vue-create.

Cromwell, V. 2016. Evan You. Interview with Evan You, the creator of Vue.js. Accessed on 23 December 2020. Retrieved from https://web.archive.org/web/20170603052649/https://betweenthewires.org/2016/11/03/evan-you/.

Daityari, S. 2020. Angular vs React vs Vue: Which Framework to Choose in 2021. A blogpost comparing JavaScript frameworks. Accessed on 22 December 2020. Retrieved from https://www.codeinwp.com/blog/angular-vs-vue-vs-react/.

Data Properties and Methods. 2020. Official Vue documentation about data properties and methods. Accessed on 20 January 2021. Retrieved from https://v3.vuejs.org/guide/data-methods.html#data-properties.

Deployment. 2020. Official Vue CLI documentation about deployment. Accessed on 9 January 2021. Retrieved from https://cli.vuejs.org/guide/deployment.html.

Different History modes. N.d. Official Vue Router documentation about history modes. Accessed on 4 February 2021. Retrieved from https://next.router.vuejs.org/guide/essentials/history-mode.html.

Directives. 2020. Official Vue API reference about directives. Accessed on 27 January 2021. Retrieved from https://v3.vuejs.org/api/directives.html.

Directives. N.d. Official Vue documentation about directives. Accessed on 23 January 2021. Retrieved from https://012.vuejs.org/guide/directives.html.

Ellingwood, J. 2014. How To Use npm to Manage Node.js Packages on a Linux Server. A tutorial on how to manage Node.js packages. Accessed on 28 December 2020. Retrieved from https://www.digitalocean.com/community/tutorials/how-to-use-npm-to-manage-node-js-packages-on-a-linux-server.

Event Handling. 2020. Official Vue documentation about event handling. Accessed on 26 January 2021. Retrieved from https://v3.vuejs.org/guide/events.html.

Getters. 2021. Official Vuex documentation about getters. Accessed on 31 January 2021. Retrieved from https://next.vuex.vuejs.org/guide/getters.html.

Getting Started. 2021. Official Vuex documentation about setup. Accessed on 30 January 2021. Retrieved from https://next.vuex.vuejs.org/guide/.

Getting Started. N.d.a. Official Vue Loader documentation about setup. Accessed on 16 January 2021. Retrieved from https://vue-loader.vuejs.org/guide/.

Getting Started. N.d.b. Official Vue Router documentation about setup. Accessed on 2 February 2021. Retrieved from https://next.router.vuejs.org/guide/.

GitHub repository for Angular. 2020. GitHub repository for the JavaScript framework Angular. Accessed on 23 December 2020. Retrieved from https://github.com/angular/angular.

GitHub repository for AngularJS. 2021. GitHub repository for the JavaScript framework AngularJS. Accessed on 5 January 2021. Retrieved from https://github.com/angular/angular.js/.

GitHub repository for Node.js. 2020. GitHub repository for the JavaScript runtime Node.js. Accessed on 28 December 2020. Retrieved from https://github.com/nodejs/node.

GitHub repository for React. 2020. GitHub repository for the JavaScript framework React. Accessed on 23 December 2020. Retrieved from https://github.com/facebook/react.

GitHub repository for serve. 2021. GitHub repository for the JavaScript module serve. Accessed on 14 January 2021. Retrieved from https://github.com/vercel/serve.

GitHub repository for Vue 2. 2020. GitHub repository for the JavaScript framework Vue 2. Accessed on 23 December 2020. Retrieved from https://github.com/vuejs/vue.

GitHub repository for Vue 3. 2020. GitHub repository for the JavaScript framework Vue 3. Accessed on 23 December 2020. Retrieved from https://github.com/vuejs/vue-next.

GUI. 2018. Creating a new project with GUI of the Vue CLI. Accessed on 4 January 2021. Retrieved from https://cli.vuejs.org/ui-new-project.png.

Installation. 2020. Official Vue CLI documentation about installation. Accessed on 29 Decemeber 2020. Retrieved from https://cli.vuejs.org/guide/installation.html.

Introduction. N.d.a. Official Vue documentation. Accessed on 23 December 2020. https://vuejs.org/v2/guide/index.html.

Introduction. N.d.b. Official Vue Loader documentation. Accessed on 16 January 2021. Retrieved from https://vue-loader.vuejs.org/.

Introduction. N.d.c. Official Vue Router documentation. Accessed on 2 February 2021. Retrieved from https://next.router.vuejs.org/introduction.html.

JavaScript frameworks. N.d. JavaScript frameworks Angular, React and Vue. Accessed on 23 December 2020. Retrieved from https://miro.medium.com/max/6000/1*4OgqQRDfBtrftDNJrVHbHw.png.

Kelch, D. 2020. Three Reasons to Use Yarn in 2020 (and Beyond). A blogpost about the benefits of using Yarn. Accessed on 28 December 2020. Retrieved from https://spin.atomicobject.com/2020/03/15/why-yarn-2020/.

List Rendering. 2021. Official Vue documentation about list rendering. Accessed on 25 January 2021. Retrieved from https://v3.vuejs.org/guide/list.html.

Meet the Team. N.d. Introduction of the active core team members. Accessed on 23 December 2020. Retrieved from https://vuejs.org/v2/guide/team.html.

Muminovic, A. 2019. Alternative Package Managers For Node.js. An article about alternative package managers for Node.js. Accessed on 19 January 2021. Retrieved from https://medium.com/maestral-solutions/alternative-package-managers-for-node-js-f52805b98064.

Mutations. 2021. Official Vuex documentation about mutations. Accessed on 31 January 2021. Retrieved from https://next.vuex.vuejs.org/guide/mutations.html.

Neagoie, A. 2018. Tech Trends Showdown: React vs Angular vs Vue. A blogpost comparing JavaScript frameworks. Accessed on 23 December 2020. Retrieved from https://zerotomastery.io/blog/tech-trends-showdown-react-vs-angular-vs-vue/.

Overview. 2019. Official Vue CLI documentation. Accessed on 29 December 2020. Retrieved from https://cli.vuejs.org/.

Pile, D. 2019. Is there any harm in using NPM and Yarn in the same project?. An answer to a Stack Overflow question. Accessed on 3 February 2021. Retrieved from https://stackoverflow.com/questions/49589493/is-there-any-harm-in-using-npm-and-yarn-in-the-same-project.

Scoped CSS. N.d. Official Vue Loader documentation about scoped CSS. Accessed on 18 January 2021. Retrieved from https://vue-loader.vuejs.org/guide/scoped-css.html.

Server-Side Rendering. N.d. Official Vue documentation about server-side rendering. Accessed on 4 January 2021. Retrieved from https://vuejs.org/v2/guide/ssr.html.

Shah, K. 2020. INFOGRAPHIC: Javascript Framework Faceoff - Angular vs React vs Vue. A blogpost comparing JavaScript frameworks. Accessed on 22 December 2020.

Retrieved from https://www.thirdrocktechkno.com/blog/infographic-javascript-framework-faceoff-angular-vs-react-vs-vue/.

Single File Components. 2020. Official Vue documentation about single-file components. Accessed on 19 January 2021. Retrieved from https://v3.vuejs.org/guide/single-file-component.html#introduction.

Special Attributes. 2020. Official Vue documentation about special attributes. Accessed on 25 January 2021. Retrieved from https://v3.vuejs.org/api/special-attributes.html.

Stack Overflow Developer Survey 2020. 2020. Results of the 2020 Developer Survey conducted by Stack Overflow. Accessed on 30 December 2020. Retrieved from https://insights.stackoverflow.com/survey/2020#technology-most-loved-dreaded-and-wanted-languages-loved.

Stack Overflow Developer Survey Results 2019. 2019. Results of the 2019 Developer Survey conducted by Stack Overflow. Accessed on 23 December 2020. Retrieved from https://insights.stackoverflow.com/survey/2019.

Star history. N.d. GitHub star history of Angular, React and Vue. Accessed on 22 December 2020. Retrieved from https://star-history.t9t.io/#angular/angular&facebook/react&vuejs/vue.

State. 2021. Official Vuex documentation about state. Accessed on 31 January 2021. Retrieved from https://next.vuex.vuejs.org/guide/state.html.

Template Syntax. 2021. Official Vue documentation about template syntax. Accessed on 20 January 2021. Retrieved from https://v3.vuejs.org/guide/template-syntax.html.

TypeScript for the New Programmer. 2020. Official TypeScript documentation about learning the technology. Accessed on 30 December 2020. Retrieved from https://www.typescriptlang.org/docs/handbook/typescript-from-scratch.html.

Using Pre-Processors. N.d. Official Vue Loader documentation about pre-processors. Accessed on 21 January 2021. Retrieved from https://vue-loader.vuejs.org/guide/pre-processors.html.

v0.9.0: Animatrix. 2014. Release information for Vue v0.9.0 Animatrix. Accessed on 23 December 2020. Retrieved from https://github.com/vuejs/vue/releases/tag/v0.9.0.

v3.0.0 One Piece. 2020. Release information for Vue v3.0.0 One Piece. Accessed on 23 December 2020. Retrieved from https://github.com/vuejs/vue-next/releases/tag/v3.0.0.

Vorback, P. 2018. Download statistics for packages. Download statistics for packages angular, react and vue between December 2018 and December 2020. Accessed on 23 December 2020. Retrieved from https://npm-stat.com/charts.html?package=angular&package=react&package=vue&from=2018-12-01&to=2020-12-01.

Vue 1. N.d. Celebration illustration for Vue 1. Accessed on 23 Decemeber 2020. Retrieved from https://web.archive.org/web/20170603082526im_/https://cdn-images-1.medium.com/max/800/1*hvV2aJXo9vKsNXjYTzODWQ.png.

Vue 2. N.d. Celebration illustration for Vue 2. Accessed on 23 December 2020. Retrieved from https://web.archive.org/web/20170603082527im_/https://cdn-images-1.medium.com/max/800/1*Ea4hEK1X64TSYAu7Xlf-Kw.jpeg.

Vue 3. N.d. Celebration illustration for Vue 3. Accessed on 23 December 2020. Retrieved from https://user-images.githubusercontent.com/499550/93624428-53932780-f9ae-11ea-8d16-af949e16a09f.png.

Vue Single-File Component Spec. N.d. Official Vue Loader documentation about the specification of a single-file component. Accessed on 20 January 2021. Retrieved from https://vue-loader.vuejs.org/spec.html#intro.

Vue.js Server-Side Rendering Guide. N.d. Official Vue documentation about server-side rendering. Accessed on 31 December 2020. Retrieved from https://ssr.vuejs.org/#what-is-server-side-rendering-ssr.

What is TypeScript?. 2020. Official TypeScript documentation. Accessed on 30 December 2020. Retrieved from https://www.typescriptlang.org/.

What is Vuex?. 2021. Official Vuex documentation. Accessed on 30 January 2021. Retrieved from https://vuex.vuejs.org/.

Why did we build Visual Studio Code?. 2020. Official Visual Studio Code documentation. Accessed on 29 December 2020. Retrieved from https://code.visualstudio.com/docs/editor/whyvscode.

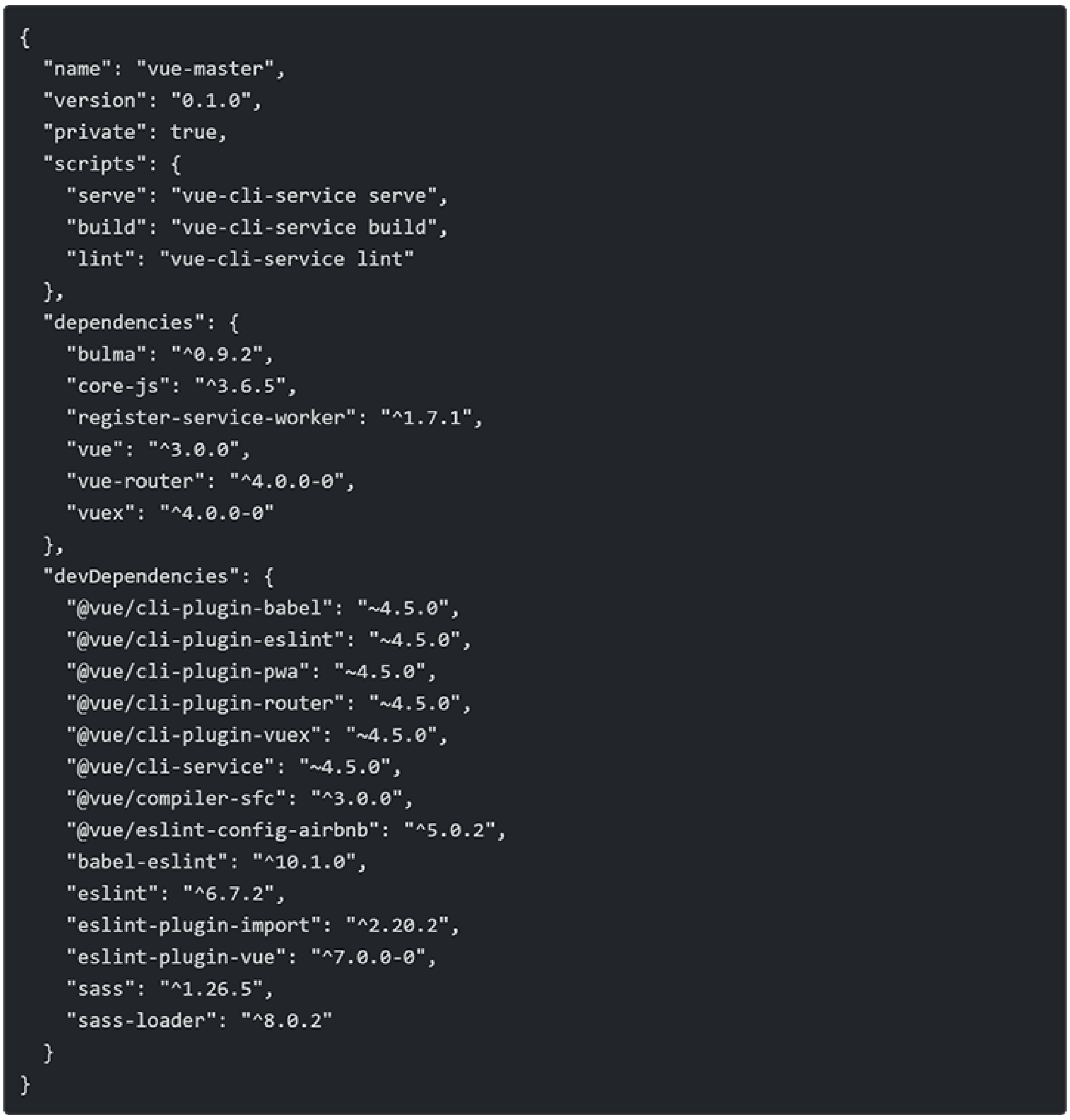
Why Quasar?. N.d. Official Quasar Framework documentation. Accessed on 4 January 2021. Retrieved from https://quasar.dev/introduction-to-quasar.

Working with CSS. 2021. Official Vue CLI documentation about CSS. Accessed on 21 January 2021. Retrieved from https://cli.vuejs.org/guide/css.html.

Wu, P. 2020. Vetur. Visual Studio Code marketplace overview of Vetur. Accessed on 29 December 2020. Retrieved from https://marketplace.visualstudio.com/items?itemName=octref.vetur.

Appendices

Appendix 1. The package.json-file of the learning tool



```json
{
  "name": "vue-master",
  "version": "0.1.0",
  "private": true,
  "scripts": {
    "serve": "vue-cli-service serve",
    "build": "vue-cli-service build",
    "lint": "vue-cli-service lint"
  },
  "dependencies": {
    "bulma": "^0.9.2",
    "core-js": "^3.6.5",
    "register-service-worker": "^1.7.1",
    "vue": "^3.0.0",
    "vue-router": "^4.0.0-0",
    "vuex": "^4.0.0-0"
  },
  "devDependencies": {
    "@vue/cli-plugin-babel": "~4.5.0",
    "@vue/cli-plugin-eslint": "~4.5.0",
    "@vue/cli-plugin-pwa": "~4.5.0",
    "@vue/cli-plugin-router": "~4.5.0",
    "@vue/cli-plugin-vuex": "~4.5.0",
    "@vue/cli-service": "~4.5.0",
    "@vue/compiler-sfc": "^3.0.0",
    "@vue/eslint-config-airbnb": "^5.0.2",
    "babel-eslint": "^10.1.0",
    "eslint": "^6.7.2",
    "eslint-plugin-import": "^2.20.2",
    "eslint-plugin-vue": "^7.0.0-0",
    "sass": "^1.26.5",
    "sass-loader": "^8.0.2"
  }
}
```