Vue.js Cheat Sheet

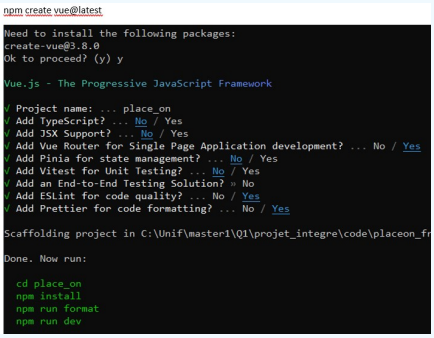
by Manon Gerrd (manongerard) via cheatography.com/195339/cs/40968/

Main ideas

Single-File Components (*.vue)	encapsulates the component's logic (JavaScript), template (HTML), and styles (CSS) in a single file
Options API	beginner-friendly by abstracting away the reactivity details and enforcing code organization via option groups

Note: there exists also the composition API

Create an app with Vite



To install npm you might need Node.js: https://nodejs.org/en
ass router, ESLint for good code quality and Prettier for code formatting
npm install : install project dependencies
npm run format : code formatting (from Prettier)
npm run dev: starts development server
vite build: create a production build

Structure

```
<script>
import ChildComp from './Chi ldC omp.vue'
export default {
  data(), computed:, methods:, mounted(), watch:,
 compon ents:, emits:, created()
    {...}
}
</script>
```

Where to write vue.js script

Structure (cont)

<template> ... </template>	Where to write the "HTML" code
<style scoped> ... </style>	Where to write the CSS code Applied to elements of the current component only

Script

`data() { return { property: value } }`	Properties returned from data() become reactive state and will be exposed on `this`
`methods: { function() { code including this.p ropert y } }`	Methods are functions that mutate state and trigger updates. They can be bound as event handlers in templates.
`mounted() { code }`	Lifecycle hooks are called at different stages of a component's lifecycle. This function will be called when the component is mounted
`computed: { fct() { return code } }`	Tracks other reactive state used in its computation as dependencies. It caches the result and automatically updates it when its dependencies change.
`watch: { prop() { code } }`	The watch callback is called when *prop* changes, and receives the new value as the argument
`compon ents: { ChildComp }`	Register child component

Script (cont)

inside child component
`props: {`
`  prop: value`
`}`
	Child component can accept input from the parent

inside child component
`emits: ['eventName'],`
`created() {`
`  this.$ emit('eventName', '*addi tional arg ')`
`}`
	Child component emits events to the parent

Directives

v-text="*prop*"	Sets the inner text
v-bind:*attr*="*prop*" or :*attr*="*prop*"	Bind an attribute to a dynamic value: reactive updates attribute
v-on:click="*fct*" or @click-="*fct*"	Listen to DOM events and execute *fct* from methods()
v-model="*prop*"	2 way data bounding: automatically syncs the form's value to the bound state
v-model.lazy="*prop*"	updates on change event
v-model.trim="*prop*"	removes extra whitespace
v-model.number="*prop*"	always return a number
v-if="*prop*" v-else, v-else-if	Render only if *property* is truthy
v-for="*element* in *array*" :key="*element.obj*" v-for="*(element, index)* in *array*" :key="*element.obj*" **array* can be replaced by a computed property	Render a list of elements based on *array* Note: always use key
v-once	Sets val once; Never update
v-show	Toggles display CSS value
v-html="*attr*"	Sets the inner HTML

Other

{{ *property* }}	Render dynamic text based on the value of *property*
ref="name"	Template ref: a reference to an element in the template
this.$refs.name	element will be exposed there Note: only accessible after the component is mounted
<childComp/> or <childComp>*slot content*</childCo-mp>	Render child component
<childComp :child-Prop="dataProp"/>	Parent can pass the prop to the child just like attributes. To pass a dynamic value, we can also use the v-bind syntax
<childComp @event-Name=(arg) => parentProp = arg"/>	Parent can listen to child-emitted events using v-on
In child template: <slot/> or <slot> *fallback content* </s-lot>	Parent can pass down template fragments to the child
<slot name='*name*'> </slot> In parent: <template v-slot:*name*> *content* </template>	Specify a name to the slot. By default, the name is 'default'

Built-in components

<Ke-epAliv-e>	Remembers the state of non-active dynamic components
<Telep-ort>	Moves an element to another place in the DOM structure
<Tr-ansiti-on>	Animates an element as it is removed from, or added to, our application with v-if or v-show, or with dynamic components.
<Tr-ansiti-onG-roup>	Animates elements that are added to our page with v-for