

# Quickly Quasar

Quasar, a Vue Widget Framework

---

[Haris Hashim](#)

© 2018 harishashim@gmail.com

## Overview

A step by step training manual style instruction to accompany mentor and mentee discussion and learning session. This is a simple introduction to Quasar.

**\*IMPORTANT\*** This document does not include explanation that is done during one to one mentorship session.



# Table of Contents

<b>Part 0: Introduction to Vue</b>	<b>2</b>
Prerequisite	2
Steps	2
Exercise	3
<b>Part 1: Getting Started</b>	<b>4</b>
Prerequisite	4
Steps	4
Exercise	4
<b>Part 2: Layout, Declaring Data, and Handling Click</b>	<b>5</b>
Prerequisite	5
Workaround	5
Steps	5

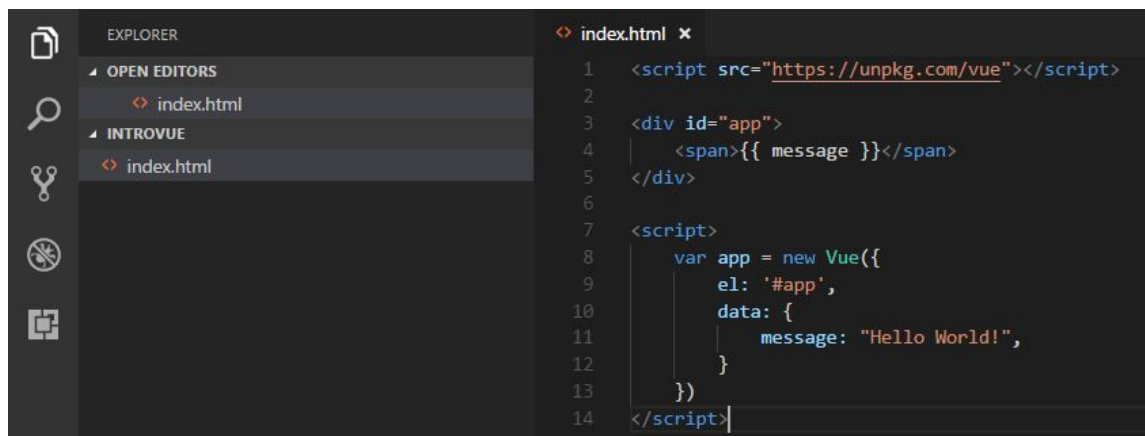
## Part 0: Introduction to Vue

### Prerequisite

1. VS Code installed
2. Chrome Browser installed
3. Internet connection

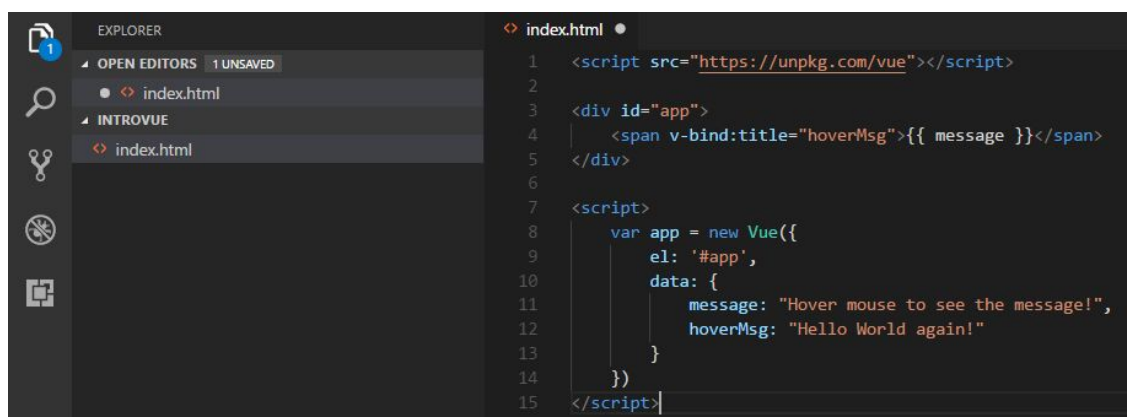
### Steps

1. Create a root folder to contain all of our project. Let's name this folder QuicklyQuasar.
2. Create another folder inside QuicklyQuasar for this the tutorial and name it introvue.
3. Open introvue folder using VS Code and Create index.html file with the following contents:



```
1 <script src="https://unpkg.com/vue"></script>
2
3 <div id="app">
4   <span>{{ message }}</span>
5 </div>
6
7 <script>
8   var app = new Vue({
9     el: '#app',
10    data: {
11      message: "Hello World!",
12    }
13  })
14 </script>
```

4. Open the file in Chrome to see the result. If the file is already opened, refresh using F5.
5. Open Chrome Dev Tool by pressing F12 key on the keyboard and open the Console tab so that you can type some JS code interactively.
6. Check the message variable value using app.message
7. Change message value by setting it to "Hi World!" instead. Observe what is reactive.
8. In above step text interpolation is used to reactively render variable value on a page. Another technique is bind element attribute. Do bellow code to implement this technique.



```
1 <script src="https://unpkg.com/vue"></script>
2
3 <div id="app">
4   <span v-bind:title="hoverMsg">{{ message }}</span>
5 </div>
6
7 <script>
8   var app = new Vue({
9     el: '#app',
10    data: {
11      message: "Hover mouse to see the message!",
12      hoverMsg: "Hello World again!"
13    }
14  })
15 </script>
```

9. Refresh Chrome and repeat step to check and change hoverMsg variable value to "Hi World again!"
10. The usage of v-bind as attribute is called **directive**. This is special attribute provided by Vue. Let's practice v-if directive by adding bellow code after the message <span> element.

```

3   <div id="app">
4     <span v-bind:title="hoverMsg">{{ message }}</span>
5     <span v-if="seen">Now you see me</span>
6   </div>
7

```

11. To show your understanding, add another variable called seen and set the value to false.
12. Save and refresh chrome, you will not see the new <span> element because seen is false. Change the value to true and observe what will happen.
13. And v-for directive to loop an element base on array variable.

```

7   <ol>
8     <li v-for="todo in todos">
9       {{ todo.text }}
10    </li>
11  </ol>

```

14. With bellow array variable in data.

```

22  todos: [
23    { text: 'Learn JavaScript' },
24    { text: 'Learn Vue' },
25    { text: 'Build something awesome' }
26  ]

```

15. Save and refresh Chrome to see the to do list.
16. Add a new to do list item by running bellow code in Chrome Developer Tools.

```
app.todos.push({ text: 'New item' })
```

17. Observe Vue reactive nature when the new item is added immediately!

## Exercise

1. This part of the training manual is a shorter version of [Introduction To Vue](#). As a take home exercise, continue with the link to learn more. Instead of redoing the introduction, adapt it to all of the above step.

## Part 1: Getting Started

### Prerequisite

1. Node JS and NPM installed

### Steps

1. Open command prompt or console in QuicklyQuasar folder as created in part 0.
2. Install Quasar command line interface by executing in command prompt

```
npm install -g quasar-cli
```

3. List all Quasar starter kit just for giggles.

```
quasar list
```

4. Install the default starter kit. This command will also create the quickly folder.

```
quasar init quickly
```

5. Follow the suggestion by changing directory to quickly folder and executing.

```
npm install
```

```
quasar dev
```

6. Open quickly folder as project workspace using VS Code and try to understand the codes.

- a. src/main.js
- b. src/router.js
- c. src/components/Hello.vue
- d. What is templates folder?

7. Execute this to relate and understand point (5.d) above

```
quasar new --list
```

8. We will replace the default Hello.vue by doing the following

- a. Create a new .vue file by doing

```
quasar new layout HelloWorld
```

- b. Change the route for path '/' to point to HelloWorld
- c. Save file to see hot reload in action.

9. Change title in HelloWorld.vue to "Hello World!", save file to see hot reload in action.

### Exercise

1. Add any type of button and properly label it as "OK". Check documentation at [this URL for button](#). And [this URL about adding component](#).
2. Do code so that when the "OK" button is pressed, title change from "Hello World!" to "Bye!".

## Part 2: Layout, Declaring Data, and Handling Click

### Prerequisite

This part is continuing part 1 and as the answer to previous exercise. Code changes for layout and script is done in `src/components/HelloWorld.vue` file.

### Workaround

There is lots of problem with eslint and code formatter provided plugin in VS Code. The only solution (at this point of time) is to disable eslint.

1. Open `build/webpack.base.conf.js`
2. Comment out the following codes

```

36   module: {
37     rules: [
38       // { // eslint
39         //   enforce: 'pre',
40         //   test: /\.vue|js$/,
41         //   loader: 'eslint-loader',
42         //   include: projectRoot,
43         //   exclude: /node_modules/,
44         //   options: {
45           //     formatter: require('eslint-friendly-formatter')
46         //   }
47       // },
48     ],
49     {
50       test: /\.js$/,
51       loader: 'babel-loader',
52       include: projectRoot,
53       exclude: /node_modules/
54     },
55   ],

```

3. If quasar dev already started, restart by doing CTRL-C and execute the command again.

### Steps

1. **Layout.** Change the layout to add a button.
  - a. Put below codes somewhere on top of `<router-view />`

```

53   <div>
54     <q-btn round color="secondary" @click="btnClick">
55       <q-icon name="card_giftcard" />
56     </q-btn>
57   </div>
58   <router-view />
59
60
61

```

- b. Take note that if saved, no button will be shown. Since q-btn and q-icon tag need to be defined first. To do that, add below codes in <script>

```

70 <script>
71   import {
72     QBtn,
73     QIcon
74   } from 'quasar'
75
76   export default {
77     components: {
78       QBtn,
79       QIcon
80     },
81     data() {

```

- c. Save and check in browser that button is shown below the title.
2. **Declaring Variable.** Rather than “hard coding” the title. We will use a variable.
- a. Add title variable to data() with previously specified value.

```

81     data() {
82       return {
83         title: 'Hello there, world!'
84       }
85     }
86   }
87 </script>

```

- b. Add the variable to the layout.

```

10   <q-toolbar-title>
11     {{title}}
12   </q-toolbar-title>
13 </q-toolbar>

```

3. **Handling Click.** Nothing will happen if button is clicked now. We need to code the handler.
- a. Add methods section and btnClick function to <script>. The code will toggle title to “Bye!” just so that we can see change and verify that it works.

```

75     data() {
76       // ...
77     },
78     methods: {
79       Complexity is 3 Everything is cool!
80       btnClick() {
81         if (this.title == 'Bye!') {
82           this.title = 'Hello there, world!'
83         } else {
84           this.title = 'Bye!'
85         }
86       }
87     }
88 </script>

```

- b. Save and test the button.