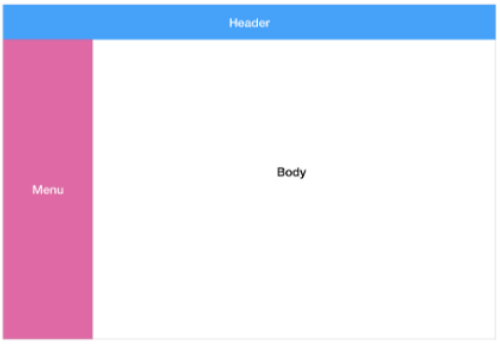
Main Reference : https://simplevue.gitbook.io/intro/

02. Component 구성해보기

1) Component : view 를 구성하는 작은 조각

- 일반적인 웹은 크게 3가지의 레이아웃으로 구성
  - 상단의 헤더 (Header.vue)
  - 왼쪽의 메뉴리스트 (Menu.vue)
  - 오른쪽의 본문 (Content.vue)



1-1) Vue 파일의 구성

- 각 vue 파일들은 자신만의 html, css, script를 가지며, component 별로 독립된 환경을 가지고 개발할 수 있음

```
<template> // html
</template>
<script> // script
export default {};
</script>
<style> // style
</style>
```

2) App.vue 파일 구성

- 이것만으로 h1의 내용인 Hello Vue 가 확인된다.

```
<template>
<div>
<h1>Hello Vue</h1>
</div>
</template>
```

- scoped 옵션을 통해 app.vue 안에 있는 h1 태그에서만의 내용의 적용은 가능, 안쓰일듯

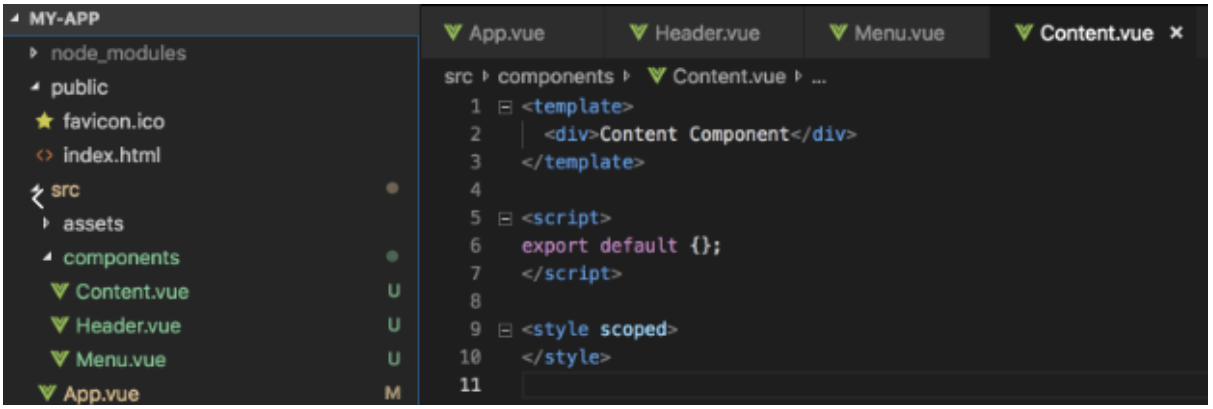
```
<style scoped>
h1 {
color: #03a9f4;
}
</style>
```

3) Component 생성

3-1) Components 폴더 생성 및 기본파일 생성

- 파일의 이름은 대문자로 시작
- Layout 을 구성하기 위해 Header.vue, Menu.vue, Content.vue 생성

```
// Header.vue, Content.vue, Menu.vue
<template>
<div> 컴포넌트 이름 </div>
</template>
<script>
export default {};
</script>
<style scoped>
</style>
```



3-2) Component 가져오기

- App.vue는 모든 조각을 조립, 다른 component 들의 부모 component 인 셈
- import 로 가져오고, export 로 (사용하도록) 내보낼 내용을 작성

```
//App.vue
<script>
import Header from "./components/Header";
import Menu from "./components/Menu";
import Content from "@/components/Content";
export default {
name: "app",
components: { // 가져온 component 들을 등록합니다.
Header,
Menu,
Content
}
};
</script>
```

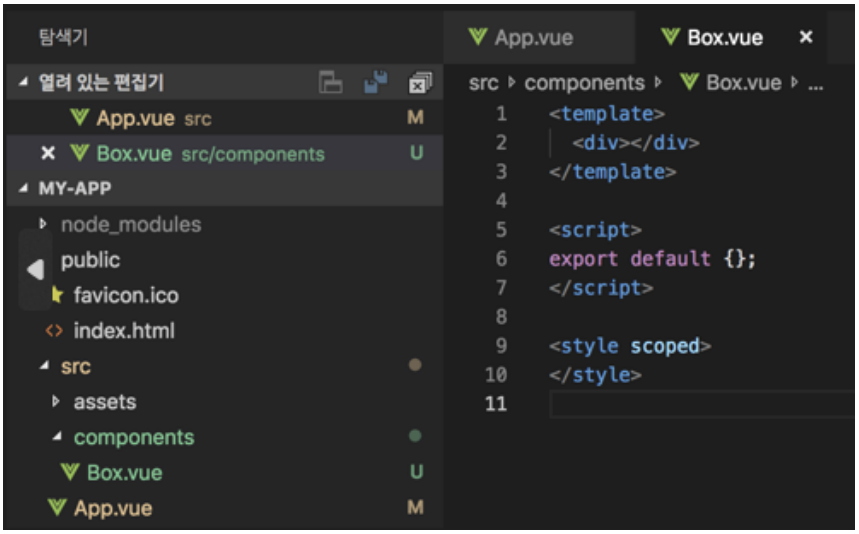
- component 를 사용하기 위해서는 script 의 export 에 작성(등록)해줘야 인식

03. State 와 Props

1) State 와 Props

- vue는 documents에 직접적인 접근을 통한 DOM 제어를 하지 않고, state를 통해 DOM 관리
- State : Style 적용을 위해 직접 데이터를 가지는 경우
- Props : 외부(내부 데이터가 아닌 곳에서 받는 경우)에서 받을 경우

2) State : 나의 데이터



Box.vue

- component의 state는 data 라는 함수를 이용해 구성

```
// Box.vue
<script>
export default {
data() { // Box 의 state
return {
width: 40, // 넓이
height: 80 // 높이
};
}
};
</script>
```

2) State를 이용해 Style 적용

- 위의 state를 바탕으로 box를 그림
- vue에 관련된 무엇인가를 적용하기 위해서는 v-bind 를 이용해야 함

2-1) Style 적용을 html tag에 인라인 적용하거나

- html tag에 인라인 적용시 v-bind:style or :style 를 이용

```
<div v-bind:style="{color: #ebebeb}"></div>
<div :style="{color: #ebebeb}"></div>
```

2-2) Box.vue에 직접 작성하거나

- box에 state 값을 바탕으로 스타일 적용

```
//Box.vue
<template>
<div v-bind:style="{ width: width + 'px', height: height + 'px' }"></div>
</template>
```

2-3) 추가 CSS

- class를 추가해 border를 추가

```
<style scoped>
.box {
border: 1px solid #000;
}
</style>
```

3) Props

- 여러개의 box를 만들려고 함, 그래서 app.vue 에 Box 여러개 추가

```
// App.vue
<template>
<div>
<Box/>
<Box/>
<Box/>
<Box/>
<Box/>
<Box/>
</div>
</template>
```

- 박스별로 색상을 다르게 주겠음, Box.vue 에 색상 class 를 추가, class가 변경됨에 따라 background 색상이 변경되도록 함

```
// Box.vue
<style scoped>
.box {
border: 1px solid #000;
}
.blue {
background: #009bff;
}
.purple {
background: #8f46ff;
}
.green {
background: #00bcac;
}</style>
```

- Box별로 색상을 다르게 주기 위한 간단한 방법은 Box component를 사용한 곳(App.vue)에서 Box component로 사용할 color class를 전달

```
// App.vue
<template>
<div>
<Box color="blue"/>
<Box color="purple"/>
<Box color="green"/>
<Box color="blue"/>
<Box color="purple"/>
<Box />
</div>
</template>
```

    - Props : Box의 입장에서는 App.vue(부모 component)에서 내려준 값(color class)

- Box.vue 에서 다음과 같이 props를 이용해 받을 수 있음
    - type를 적어줌으로써 props에 대한 안전함을 보장 가능
    - 해당 props 가 내려오지 않을 경우를 방지하기 위해 default 값 또한 줄 수 있음

```
// Box.vue
<script>
export default {
props: {
color: { type: String, default: "" }
},
data() {
return {
width: 40,
height: 80
};
}
};
</script>
```

3-1) Props를 바탕으로 class bind

- App.vue 에서 내려주는 color 값을 받을 준비는 된 것, 이제 받은 값을 Box component에 class 로 적용해야 함
- props나 state를 바탕으로 class를 적용해주기 위해 v-bind:class or :class 사용해야함

```
<div v-bind:class="[state, props]"></div>
<div :class="[state, props]"></div>
```

- class를 Box에 적용

```
<template>
<div
v-bind:class="['box', color]"
v-bind:style="{ width: width + 'px', height: height + 'px' }">
</div>
</template>
```

04. v-for 를 이용한 리스트 렌더링

1) 데이터 준비

- component를 구성할 webtoon 데이터 준비

```
// App.vue
export default {
data() {
return {
webtoons: [
{
name: "햄스터와 그녀",
link: "http://webtoon.daum.net/webtoon/view/hamsterandher",
img:"http://t1.daumcdn.net/webtoon/op/478cdf37f585607982ffa9e35b432e8503be8a54"},
{
name: "프롬 스타",
link: "http://webtoon.daum.net/webtoon/view/fromstar",
img:"http://t1.daumcdn.net/webtoon/op/a7fb953d722c1130bfc18440f7e3ce448ece57a1"
},
{
name: "위대한 로맨스",
link: "http://webtoon.daum.net/webtoon/view/greatromance",
img:"http://t1.daumcdn.net/webtoon/op/a816281cb4df5c50a20ac386fd6e496643d0f085"
},
{
name: "빛나는 손을",
link: "http://webtoon.daum.net/webtoon/view/Hand",
img: "http://t1.daumcdn.net/cartoon/5913FCAC0234C50001"}]};}};
```

2) Component 준비

- webtoon.vue 의 component 생성, webtoon component는 webtoon 데이터를 props로 받음

```
// Webtoon.vue
<template>
<div>
<h2>Webtoon</h2>
<ul class="wrap"></ul>
</div>
</template>
<script>
export default {
props: {
items: { type: Array, default: () => [] }
}
};
</script>
<style>
</style>
```

- App.vue 에서 component 를 불러오고 props 를 전달해줍니다.
- App.vue 의 state (webtoons) 를 webtoon 에게 props 로 데이터를 넘겨줍니다.

```
// App.vue
<template>
<div id="app">
<Webtoon :items="webtoons"/>
</div>
</template>
<script>
import Webtoon from "./components/Webtoon";
export default {
components: {
Webtoon
}
// data .....
}
</script>
```

3) Loop를 이용한 component 구성

- v-for 라는 디렉티브를 이용하여 리스트를 렌더링 할 수 있습니다.
- 리스트 렌더링 되는 component 는 항상 key 라는 props 값이 필요합니다.
- 리스트 렌더링시에는 항상 **key** 라는 값이 필요 **:** 가상돔에서 리스트 component 에서 변경된 부분을 감지할 때 저 key 라는 값을 이용하기 때문에 합니다.

```
<div v-for="i in 10" :key={i}>{i}</div>
```

4) webtoon component 를 구성

- v-for 를 이용하여 item 리스트를 렌더링 합니다. (item: 객체, idx: 배열의 인덱스), key 값은 인덱스 값을 이용합니다.
- **:** 가 붙으면 **" "** 영역이 **JS** 영역으로 변경 : :href , :src 같이 기본 태그에 : 가 붙는다. 이로 인해 JS 로 바뀌어 item 객체에 접근할 수 있는 것 입니다.

```
// webtoon.vue
<template>
  <div>
   <h2>Webtoon</h2>
   <ul class="wrap">
          <li class="item" v-for="(item, idx) in items" :key="{idx}"> //이 구문에 대한 설명 필요
           <a :href="item.link" target="_blank">
             <img :src="item.img">
             <span class="tit">제목: {{item.name}}</span>
          </a>
   </li>
  </ul>
 </div>
</template>
```

5) style 적용

Webtoon component 에 style 적용

```
// Webtoon.vue
<style scoped>
h2 {
text-align: center;
}
a {
list-style: none;
text-decoration: none;
}
li {
list-style: none;
}
.wrap {
max-width: 450px;
width: 100%;
margin: 0 auto;
}
.item {
border-bottom: 1px solid #ebebeb;
margin-bottom: 25px;
}
.tit {
display: inline-block;
font-size: 18px;
font-weight: bold;
color: #000;
padding: 20px 15px;
}
img {
width: 100%;
background: #ebebeb;
border-radius: 4px;
}
</style>
```

05. v-if, v-show를 이용한 조건부 렌더링

세부기능 이므로 일시적 생략

06. Data Binding

1) 진행 과정 정리

- 단방향 데이터 바인딩 : 앞선 과정은 양쪽 (vue 인스턴스와 component) 에서 접근을 하는 것이 아닌 Vue 인스턴스의 값을 component 에게 준것 뿐
- 양방향 데이터 바인딩 : Vue 인스턴스와 component 가 서로의 데이터에 접근하는 것을 말합니다. vue 에서는 v-model 디렉티브를 이용하여 양방향 데이터 바이딩을 지원합니다.

2) input

- `v-model` 은 state 값을 꼭 사용해야합니다. props 로 내려받은 값을 바로 `v-model` 에서 사용할 수는 없습니다.

```
// App.vue
<template>
<div>
<h1>{{ title }}</h1>
<input v-model="title">
</div>
</template>
<script>
export default {
data() {
return {
title: ""};}};
</script>
<style>
</style>
```

  - state 를 이용하여 component 를 구성하게된다면 jQuery 를 이용하여 개발하던 방식처럼 직접적인 dom의 접근을 피할 수 있습니다. 어떤 값을 변경하고 표시하기 위해 변경된 값을 가져오고, 표시해줄 dom 을 잡아와서 innerText 와 같은 함수를 이용하여 텍스트를 넣어주는 식의 dom 접근 개발방식에서 벗어날 수 있습니다.
  - 작성하거나 전달하는 과정이 상당히 빠르게 진행될 수 있다.

데이터 바인딩 : JS 데이터를 HTML에 꽂아넣는 문법

```
export default {
 data(){
   return{
     price1 :60,
     price2 :70
 }, }
```

등을 저장해두고 호출
필요시 <template> 에서 {{ price1 }} 로 호출

- HTML에 그대로 작성하면 추후 데이터 변경 어려움,
- Vue의 실시간 자동 렌더링을 사용, (export 의 데이터의 변경사항이 자동으로 반영됨)
  - 자주 변할 데이터를 보관해 다른 계산에도 활용

08. Router

1) router.js 추가

- path 마다 다른 component 를 보여주기 위해서는 설정이 필요
- path 별로 보여줄 component 설정을 추가시

```
import Vue from "vue";
import VueRouter from "vue-router";
import ErrorPage from "./views/ErrorPage";
import Home from "./views/Home";

Vue.use(VueRouter);

const router = new VueRouter({
  mode: "history",
  routes: [
    { path: "/", component: Home },
    {
      path: "*",
      component: ErrorPage
    }
  ]
});

export default router;
```

  - Vue.use : vue 에서 vue router 를 사용하기 위해 알려줘야 함
  - mode: “history” : browser history mode를 사용
  - routes : path 별 component 추가

- pageRouter.js (강의)

```
const pageRouter = { //이곳에서의 경로지정을 통해 연결, 화면에 띄우겠다.
  path: "/",
  name: "layout",
  redirect: "/main", //첫화면에는 main의 내용을 뿌려라
  component: () => import("@/layout/index.vue"),
  children: [ //기본은 index.vue를 적용되지만, 특정 경로도 지정시 children으로
    {
      path: "/main",
      name: "main",
      component: () => import("@/views/main.vue"),
    },],};
export default pageRouter;
```

2) Home.vue 추가 (메인 첫 화면) / Error Page 추가 (미정의된 paht로 진입시)

```vue
// src/views/Home.vue

<template>
  <div>
    <h1>Home Page</h1>
  </div>
</template>

<script>
export default {};
</script>

<style>
</style>
```

```vue
// src/views/ErrorPage.vue

<template>
  <div>
    <h1>Error Page</h1>
  </div>
</template>

<script>
export default {};
</script>

<style>
</style>
```

- index.vue (강의)

```vue
<template>
 <section id="wrap">
  <h1 class="blind">웹사이트 제목</h1>
  <Header /> <!--헤더의 영역을 지정해줘야 함-->
  <section id="container">
    <router-view /> <!--container에 연결된 애를 화면에 출력해줘라-->
  </section>
  <Footer />
 </section>
</template>

<script>
import Header from "@/layout/components/Header.vue";
import Footer from "@/layout/components/Footer.vue"; //import로 가져오고
export default { //가져온것을 적용
 components:{
  Header,
  Footer,
 },
};
</script>
```

3) main.js 에 router 추가

- router.js 를 추가만 한다고 router 를 이용할 수 있는 것은 아닙니다.
- main.js 에 추가를 해줘야합니다.

```
import Vue from "vue";
import App from "./App.vue";
import router from "./router";

Vue.config.productionTip = false;

new Vue({
  router,
  render: h => h(App)
}).$mount("#app");
```

- main.js(강의)

```
import Vue from 'vue'
import App from './App.vue'
import router from './router'; //이 부분을 의미하는 듯
import './common/plugins/bootstrap-vue'
import './common/plugins/vue-slick'
Vue.config.productionTip = false

new Vue({
 router,
 render: h => h(App)
}).$mount('#app')
```

4) App.vue 에 router-view 추가

- routing 된 component 를 보여주기 위해서는 App.vue 에 router-view 를 추가해줘야합니다.

```
<template>
  <div>
    <router-view/>
  </div>
</template>

<script>
export default {};
</script>

<style>
</style>
```

- App.vue (강의)

```
<template>
 <div id="appPage">
   <router-view/>
 </div>
</template>
```