원쌤의 Vue.js 퀵스타트

8. 컴포넌트 심화

1. 단일 파일 컴포넌트에서의 스타일

❖이전까지 학습한 내용에서의 스타일 적용

- 모두 전역 스타일로 사용됨
  - 단일 파일 컴포넌트 내의 〈style /〉 태그 이용
  - css 파일을 직접 import
- 만일 여러 스타일에서 동일한 이름의 CSS 클래스명을 사용한다면?
  - 충돌발생
  - 마지막에 로딩한 스타일이 나타남
- 충돌을 피하기 위한 방법
  - 범위 CSS
  - CSS 모듈

❖새로운 프로젝트 생성

```
npm init vue style-test
cd style-test
npm install
```

src/components 디렉토리 내의 파일 삭제
src/components에 Child1.vue, Child2.vue, Child3.vue 생성

1.1 범위 CSS

❖충돌 확인

- 예제 08-01 : src/components/Child1.vue, 같은 방식으로 Child2.vue, Child3.vue 작성
  - Child2는 blue, Child3은 orange

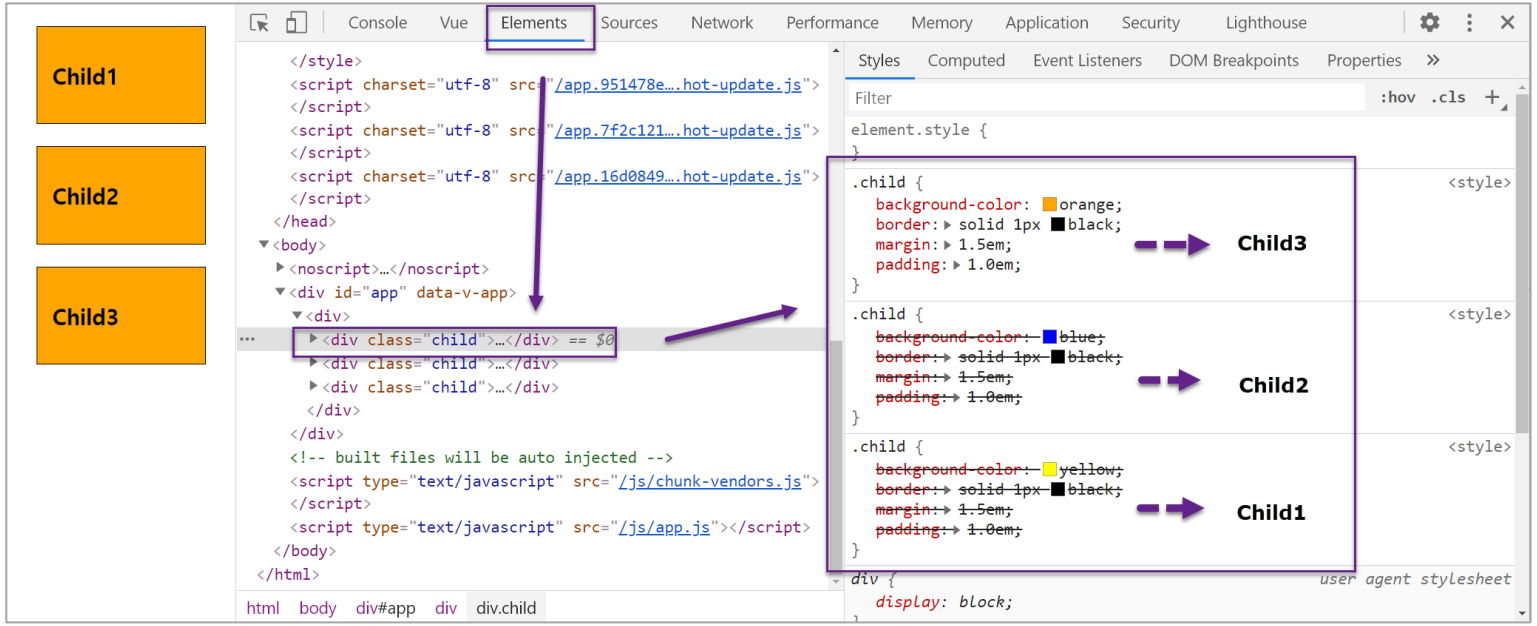
```
1   <template>
2     <div class="child">
3       <h2>Child1</h2>
4     </div>
5   </template>
6
7   <script>
8   export default {
9     name : "Child1"
10  }
11  </script>
12
13  <style>
14  .child { background-color: yellow; border:solid 1px black; margin:1.5em; padding: 1.0em;  }
15  </style>
```

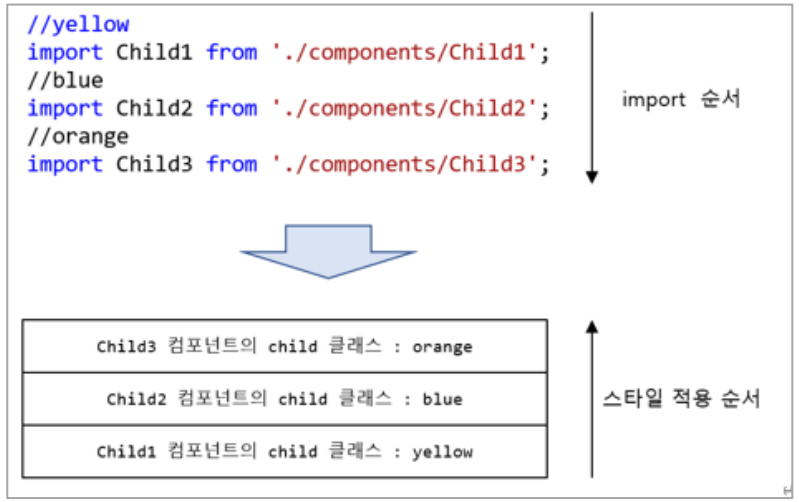
1.1 범위 CSS

- 예제 08-02 : src/App.vue 변경

```
1  <template>
2    <div>
3      <Child1 />
4      <Child2 />
5      <Child3 />
6    </div>
7  </template>
8
9  <script>
10 import Child1 from './components/Child1.vue';
11 import Child2 from './components/Child2.vue';
12 import Child3 from './components/Child3.vue';
13
14 export default {
15   name : "App",
16   components : { Child1, Child2, Child3 }
17 }
18 </script>
```

- 실행 결과 충돌 확인

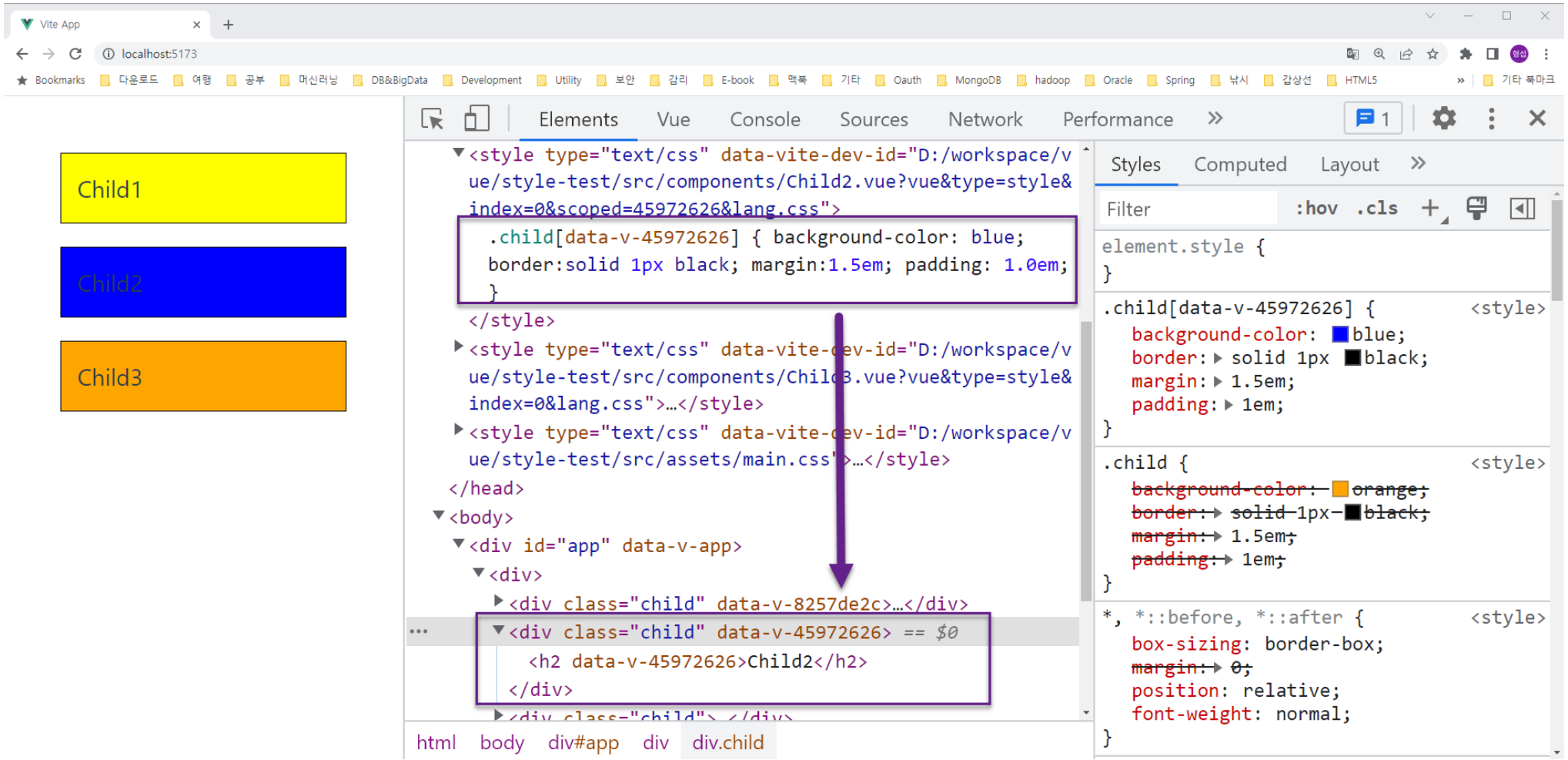




1.1 범위 CSS

❖ 범위 CSS 적용

- ⟨style⟩ ---⟩ ⟨style scoped⟩ : data-v-xxxxxx 특성으로 구분



1.1 범위 CSS

일반적으로 기본 밑바탕이 되는 CSS 클래스들을 css 파일로 작성하여 import './main.css' 와 같이 src/main.js에서 import합니다. 가장 먼저 import하기 때문에 앱 전체의 공통 스타일을 적용하기에 편리합니다. 그리고 각 컴포넌트에 적용할 CSS 클래스들은 .vue 파일에 범위 css로 충돌을 피해 적용하는 것이 바람직합니다.

1.2 CSS 모듈

❖CSS 모듈이란?

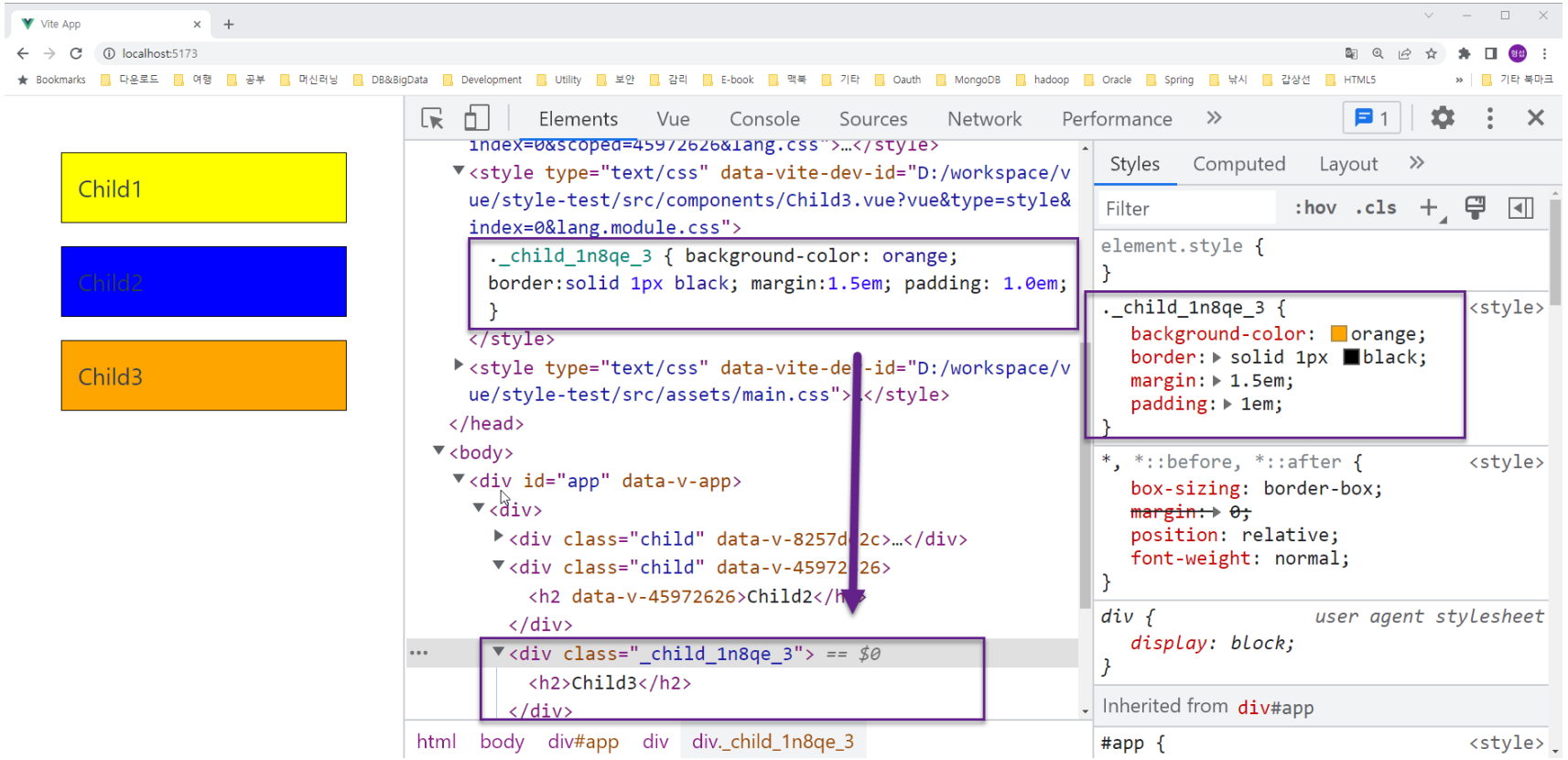
- CSS 스타일을 마치 객체처럼 다룰 수 있도록 함
- ⟨style module⟩

❖예제 08-03 : src/components/Child3.vue 변경

```
1   <template>
2     <div :class="$style.child">
3       <h2>Child3</h2>
4     </div>
5   </template>
6
7   <script>
8   export default {
9     name : "Child3",
10    created() {
11      console.log(this.$style)
12    }
13  }
14  </script>
15
16  <style module>
17  .child { background-color: orange; border:solid 1px black; margin:1.5em; padding: 1.0em;  }
18  </style>
```

1.2 CSS 모듈

❖CSS 모듈 적용 결과



▼{child: '_child_1n8qe_3'}
child: "_child_1n8qe_3"
▶[[Prototype]]: Object

```
<div :class="[$style.child, $style.italic]"> ..... </div>
```

2. 슬롯

❖ 이전에 학습한 속성(props)

- 정보 전달 방향 : 부모 -〉 자식
- 부모-〉자식으로 템플릿 정보를 전달해야 한다면?

❖슬롯(Slot)

- 부모 컴포넌트에서 자식 컴포넌트로 템플릿 정보를 전달하는 방법을 제공함

2.1 슬롯 사용 전의 컴포넌트

❖새로운 프로젝트 생성

```
npm init vue slot-test
cd slot-test
npm install
```

❖예제 08-04 : src/components/CheckBox1.vue

```
1   <template>
2     <div>
3       <input type="checkbox" :value="id" :checked="checked"
4        @change="$emit('check-changed', {id, checked: $event.target.checked })" />
5       <span v-if="checked === true" style="color:blue; text-decoration:underline;">
6         <i>{{label}}</i>
7       </span>
8       <span v-else style="color:gray">{{label}}</span>
9     </div>
10  </template>
11
12  <script>
13  export default {
14    name : "CheckBox1",
15    props : ["id", "checked", "label"]
16  }
17  </script>
```

2.1 슬롯 사용 전의 컴포넌트

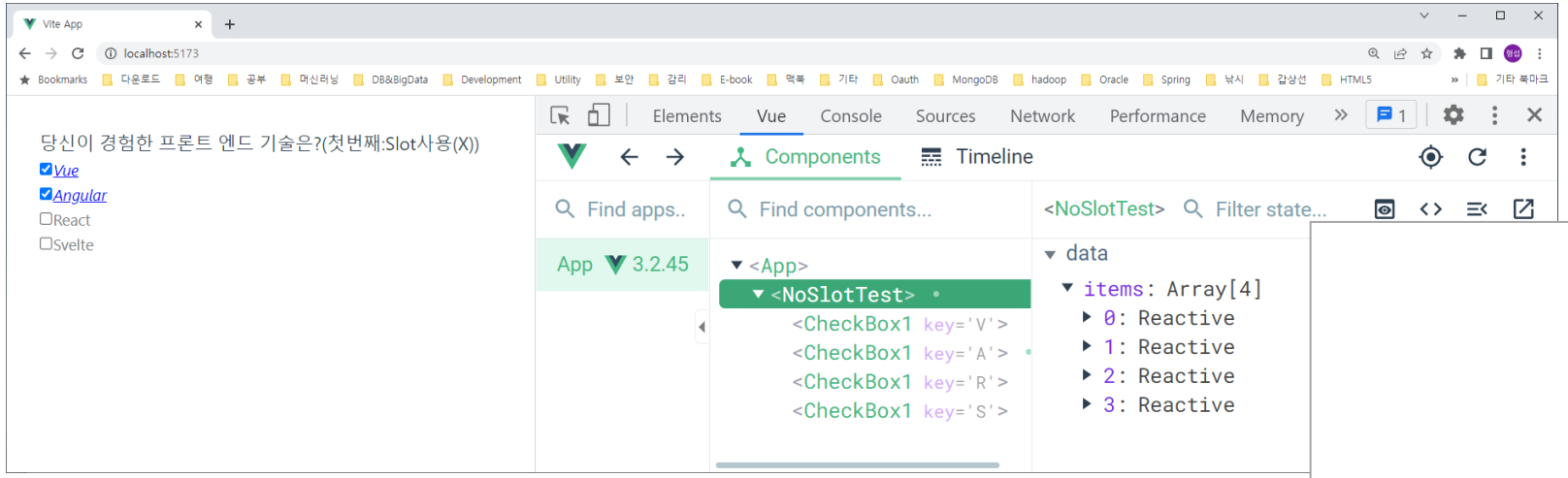
❖예제 08-05~06 : src/components/NoSlotTest.vue, src/App.vue

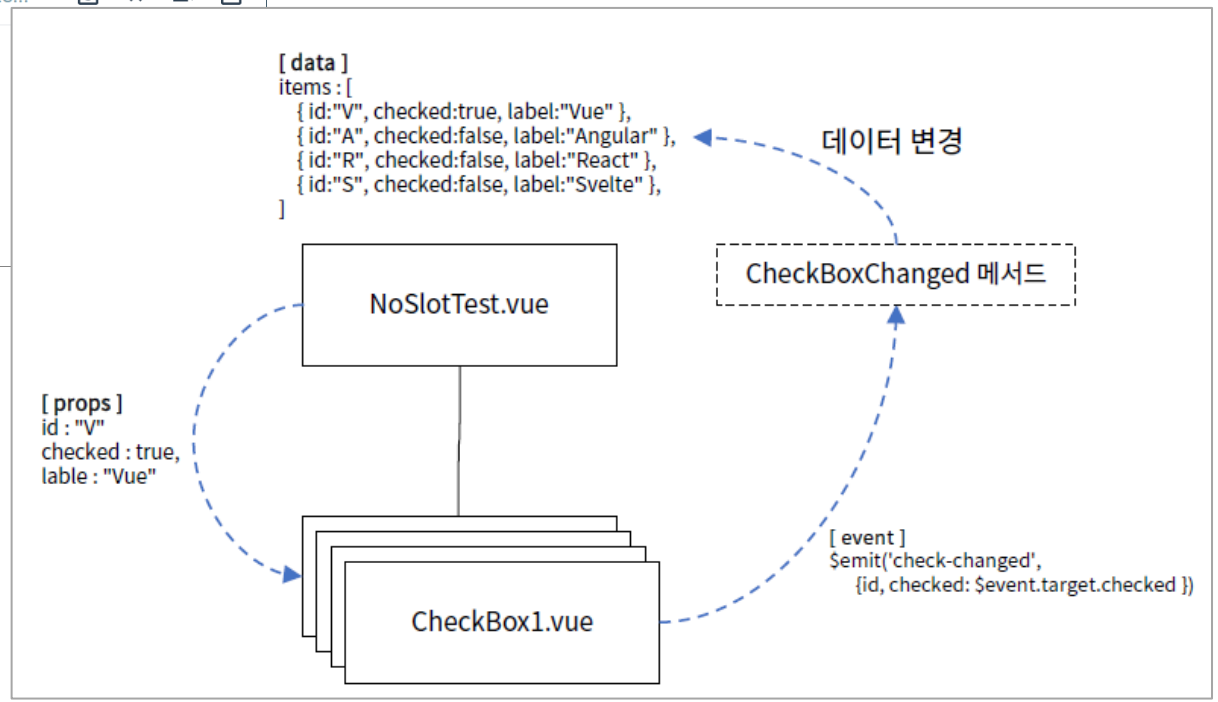
```vue
<template>
  <div>
    <h3>당신이 경험한 프론트 엔드 기술은?(첫번째:Slot사용(X))</h3>
    <CheckBox1 v-for="item in items" :key="item.id" :id="item.id" :label="item.label"
      :checked="item.checked" @check-changed="CheckBoxChanged">
    </CheckBox1>
  </div>
</template>

<script>
import CheckBox1 from './CheckBox1.vue';

export default {
  name : "NoSlotTest",
  components : { CheckBox1 },
  data() {
    return {
      items : [
        { id:"V", checked:true, label:"Vue" },
        { id:"A", checked:false, label:"Angular" },
        { id:"R", checked:false, label:"React" },
        { id:"S", checked:false, label:"Svelte" },
      ]
    }
  },
  methods : {
    CheckBoxChanged(e) {
      let item = this.items.find((item)=> item.id === e.id);
      item.checked = e.checked;
    }
  }
}
</script>
```

```vue
<template>
  <div>
    <NoSlotTest />
  </div>
</template>

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

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

2.1 슬롯 사용 전의 컴포넌트

❖ 실행 결과와 예제 구조





- 실행은 잘되지만
  - CheckBox1 컴포넌트의 재사용성과 관련해 불편함
  - 템플릿이 정해져 있음
    - 템플릿을 변경하고 싶다면 컴포넌트 자체를 수정해야 함

2.2 슬롯의 기본 사용법

❖슬롯

- 부모 컴포넌트에서 자식 컴포넌트로 템플릿을 전달하는 방법을 제공

❖예제 변경

- CheckBox 컴포넌트 변경
  - 부모 컴포넌트로부터 라벨을 포함한 템플릿을 전달받도록...
- CheckBox2.vue, SlotTest.vue 컴포넌트 추가

❖예제 08-07 : src/components/CheckBox2.vue

```
1  <template>
2    <div>
3      <input type="checkbox" :value="id" :checked="checked"
4          @change="$emit('check-changed', {id, checked: $event.target.checked })" />
5      <slot>Item</slot>
6    </div>
7  </template>
8
9  <script>
10 export default {
11   name : "CheckBox2",
12   props : ["id", "checked"]
13 }
14 </script>
```

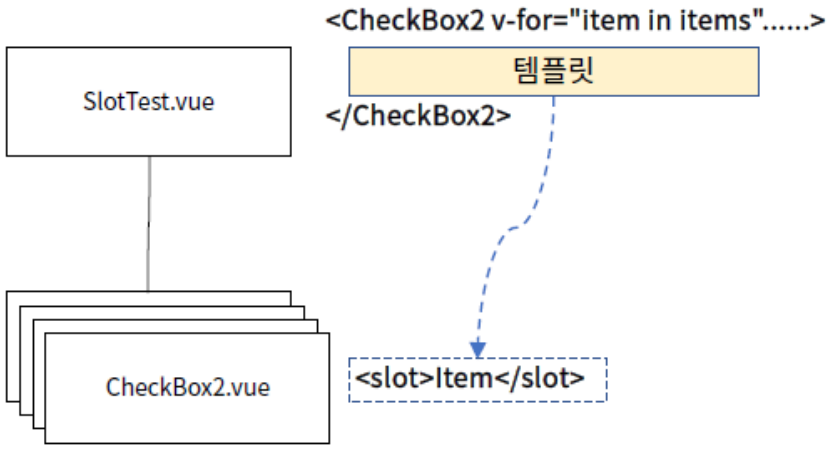
2.2 슬롯의 기본 사용법

❖예제 08-08: src/components/SlotTest.vue

```
<template>
  <div>
    <h3>당신이 경험한 프론트 엔드 기술은?(두번째:slot기본)</h3>
    <CheckBox2 v-for="item in items" :key="item.id" :id="item.id"
      :checked="item.checked" @check-changed="CheckBoxChanged">
      <span v-if="item.checked" style="color:blue; text-decoration:underline;">
        <i>{{item.label}}</i></span>
      <span v-else style="color:gray">{{item.label}}</span>
    </CheckBox2>
  </div>
</template>

<script>
import CheckBox2 from './CheckBox2.vue';

export default {
  name : "SlotTest",
  components : { CheckBox2 },
  data() {
    return {
      items : [
        { id:"V", checked:true, label:"Vue" },
        { id:"A", checked:false, label:"Angular" },
        { id:"R", checked:false, label:"React" },
        { id:"S", checked:false, label:"Svelte" },
      ]
    }
  },
  methods : {
    CheckBoxChanged(e) {
      let item = this.items.find((item)=> item.id === e.id);
      item.checked = e.checked;
    }
  }
}
</script>
```

2.2 슬롯의 기본 사용법

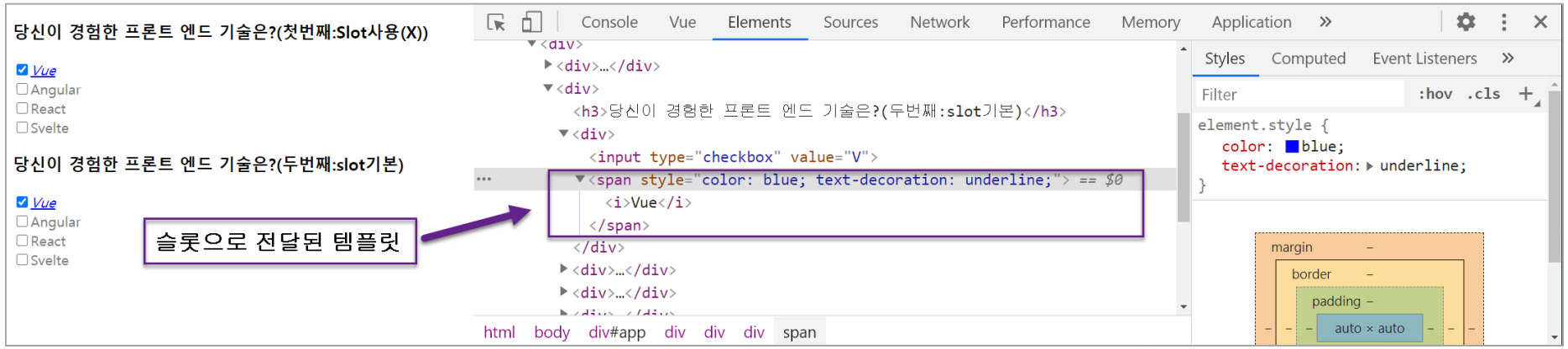
❖ 예제 08-07~08 구조



전달되는 템플릿이 없는 경우 보여줄 fallback UI를
<slot></slot> 사이에 작성함

❖ 예제 08-09: src/App.vue 변경

```
1   <template>
2     <div>
3       <NoSlotTest />
4       <SlotTest />
5     </div>
6   </template>
7
8   <script>
9   import NoSlotTest from './components/NoSlotTest.vue'
10  import SlotTest from './components/SlotTest.vue'
11
12  export default {
13    name: 'App',
14    components: { NoSlotTest, SlotTest }
15  }
16  </script>
```



2.3 명명된 슬롯

❖이전 예제는 단하나의 슬롯을 사용
- 만일 여러개의 슬롯을 사용하고 싶다면?
- 명명된 슬롯(Named Slot)을 사용해야 함

❖예제 변경
- 체크박스 앞에 이모지(Emoji)를 추가할 수 있도록 기능 변경
  - 어떤 이모지를 보여줄지는 부모 컴포넌트에서 지정할 수 있도록 슬롯 추가
- 슬롯이 두개

❖예제 08-10 : src/components/CheckBox3.vue

```
1  <template>
2    <div>
3      <slot name="icon"></slot>
4      <input type="checkbox" :value="id" :checked="checked"
5          @change="$emit('check-changed', {id, checked: $event.target.checked })" />
6      <slot name="label">Item</slot>
7    </div>
8  </template>
9  ......
```

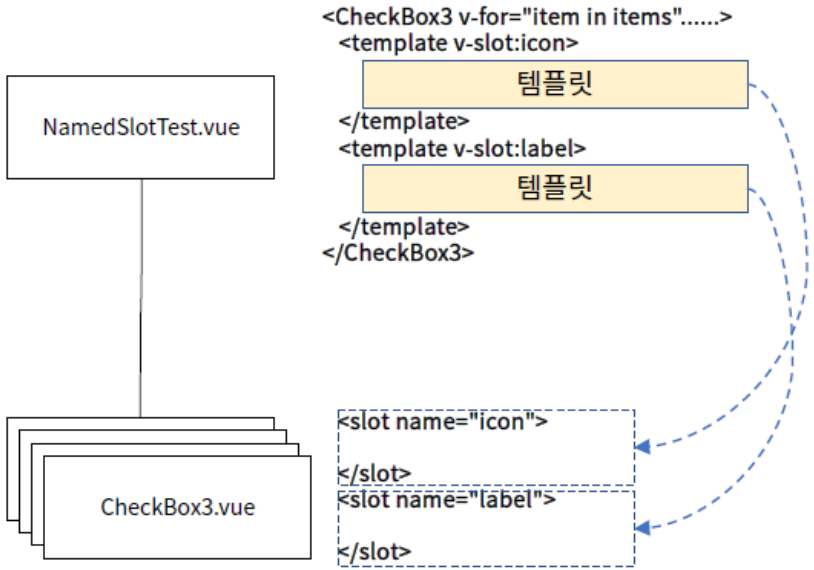
2.3 명명된 슬롯

❖예제 08-11 : src/components/NamedSlotTest.vue

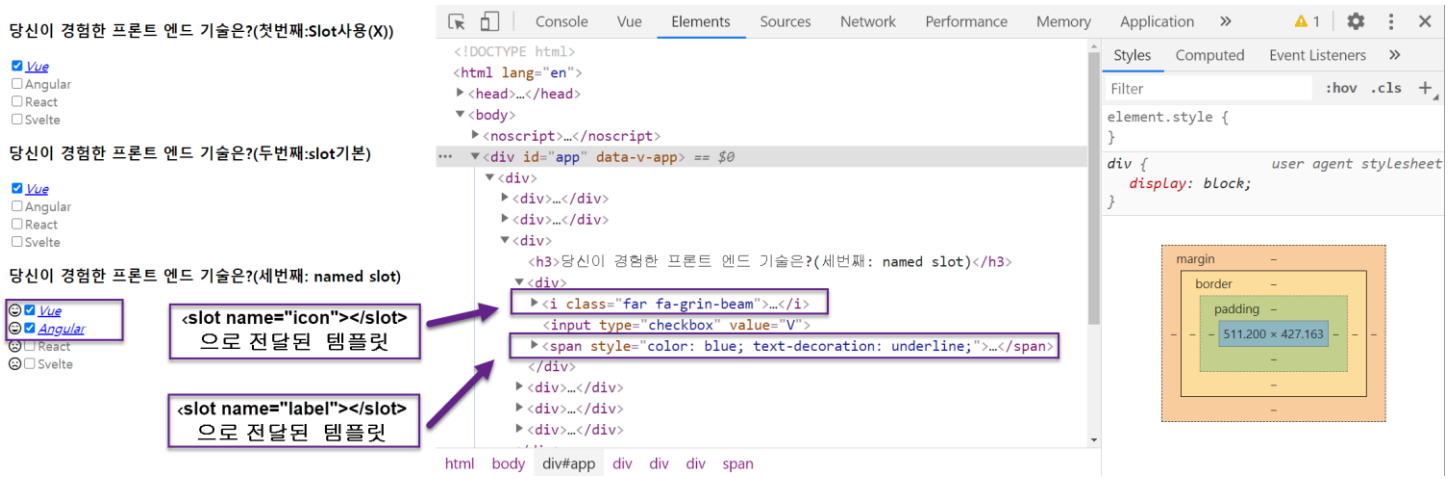
```vue
<template>
  <div>
    <h3>당신이 경험한 프론트 엔드 기술은?(세번째: named slot)</h3>
    <CheckBox3 v-for="item in items" :key="item.id"
      :id="item.id" :checked="item.checked"
      @check-changed="CheckBoxChanged">
      <template v-slot:icon>
        <i v-if="item.checked" class="far fa-grin-beam"></i>
        <i v-else class="far fa-frown"></i>
      </template>
      <template v-slot:label>
       <span v-if="item.checked" style="color:blue; text-decoration:underline;">
          <i>{{item.label}}</i></span>
       <span v-else style="color:gray">{{item.label}}</span>
      </template>
    </CheckBox3>
  </div>
</template>

<script>
import CheckBox3 from './CheckBox3.vue';

export default {
  name : "SlotTest",
  components : { CheckBox3 },
  data() {
    return {
      items : [
        { id:"V", checked:true, label:"Vue" },
        { id:"A", checked:false, label:"Angular" },
        { id:"R", checked:false, label:"React" },
        { id:"S", checked:false, label:"Svelte" },
      ]
    }
  },
  methods : {
    CheckBoxChanged(e) {
      let item = this.items.find((item)=> item.id === e.id);
      item.checked = e.checked;
    }
  }
}
</script>
<style>
@import url("https://cdnjs.cloudflare.com/ajax/libs/font-awesome/5.14.0/css/all.min.css");
</style>
```

2.3 명명된 슬롯

❖예제 08-10~11 구조




```vue
<template>
  <div>
    <NoSlotTest />
    <SlotTest />
    <NamedSlotTest />
  </div>
</template>

<script>
import NoSlotTest from './components/NoSlotTest.vue'
import SlotTest from './components/SlotTest.vue'
import NamedSlotTest from './components/NamedSlotTest.vue'

export default {
  name: 'App',
  components: { NoSlotTest, SlotTest, NamedSlotTest }
}
</script>
```



2.3 명명된 슬롯

❖명명된 슬롯을 사용한 레이아웃 설정

- 예제 08-13 : src/component/Layout.vue

```
<template>
    <div class="wrapper">
        <header class="box header">
            <slot name="header"></slot>
        </header>
        <aside class="box sidebar">
            <slot name="sidebar"></slot>
        </aside>
        <section class="box content">
            <slot></slot>
        </section>
        <footer class="box footer">
            <slot name="footer"></slot>
        </footer>
    </div>
</template>
<script>
export default {
    name : "Layout"
}
</script>
<style scoped>
……(생략)
</style>
```

2.3 명명된 슬롯

❖명명된 슬롯을 사용한 레이아웃 설정

- 예제 08-14 : src/App2.vue

```
<template>
  <Layout>
    <template v-slot:header>
      <h2>헤더 영역</h2>
    </template>
    <template v-slot:sidebar>
      <h3>사이드</h3>
      <h3>사이드</h3>
      <h3>사이드</h3>
    </template>
    <template v-slot:default>
      <h1>컨텐트 영역</h1><h1>컨텐트 영역</h1><h1>컨텐트 영역</h1>
      <h1>컨텐트 영역</h1><h1>컨텐트 영역</h1><h1>컨텐트 영역</h1>
    </template>
    <template v-slot:footer>
      <h2>Footer text</h2>
    </template>
  </Layout>
</template>

<script>
import Layout from './components/Layout.vue';
export default {
  name : "App2",
  components : { Layout }
}
</script>
```

❖명명된 슬롯을 사용한 레이아웃 설정

- 예제 08-15 : src/main.js 변경

```
import { createApp } from 'vue'
//import App from './App.vue'
import App from './App2.vue'

import './assets/main.css'

createApp(App).mount('#app')
```

2.3 명명된 슬롯

❖ 예제 08-13~15 실행 결과

2.3 명명된 슬롯

❖만일 Layout을 변경하고 싶다면?

- Layout 컴포넌트에서 패널의 배치만 변경하면 됨.

예제 08-16 : src/components/Layout.vue 변경

```
<style>
......(생략)
.wrapper {
    display: grid;
    grid-gap: 5px;
    grid-template-rows : 100px 1fr 100px;
    grid-template-columns: 1fr 200px;
    grid-template-areas:
        "header  header"
        "content sidebar"
        "footer  footer";
    background-color: #fff;
    color: #444;
}
......(생략)
</style>
```

2.4 범위 슬롯

❖지금까지의 슬롯

- 부모에서 자식으로 템플릿을 전달함
  - 템플릿에 바인딩되는 데이터는 부모 컴포넌트의 데이터, 속성임
- 부모 컴포넌트에서 템플릿에 데이터를 바인딩할 때 자식 컴포넌트의 데이터를 이용하려면?
  - 범위 슬롯을 사용해야 함.

❖범위 슬롯

- 마치 속성을 전달하듯이 자식에서 부모로 데이터를 전달
- 전달한 데이터는 슬롯 템플릿(〈template〉) 내부 범위에서만 사용 --〉 범위 슬롯
  - 렌더링할 용도로만 사용

2.4 범위 슬롯

❖예제 작성전 그림 먼저 보기

2.4 범위 슬롯

❖예제 08-17 : src/components/CheckBoxList.vue

```
1   <template>
2     <div v-for="item in items" :key="item.id">
3       <slot name="icon" :checked="item.checked"></slot>
4       <input type="checkbox" :value="item.id" :checked="item.checked"
5     @change="$emit('check-changed', {id:item.id, checked: $event.target.checked })" />
6       <slot :checked="item.checked" :label="item.label"></slot>
7     </div>
8   </template>
9
10  <script>
11  export default {
12    name : "CheckBoxList",
13    props : ["items"]
14  }
15  </script>
```

2.4 범위 슬롯

❖예제 08-18 : src/components/ScopedSlotTest.vue

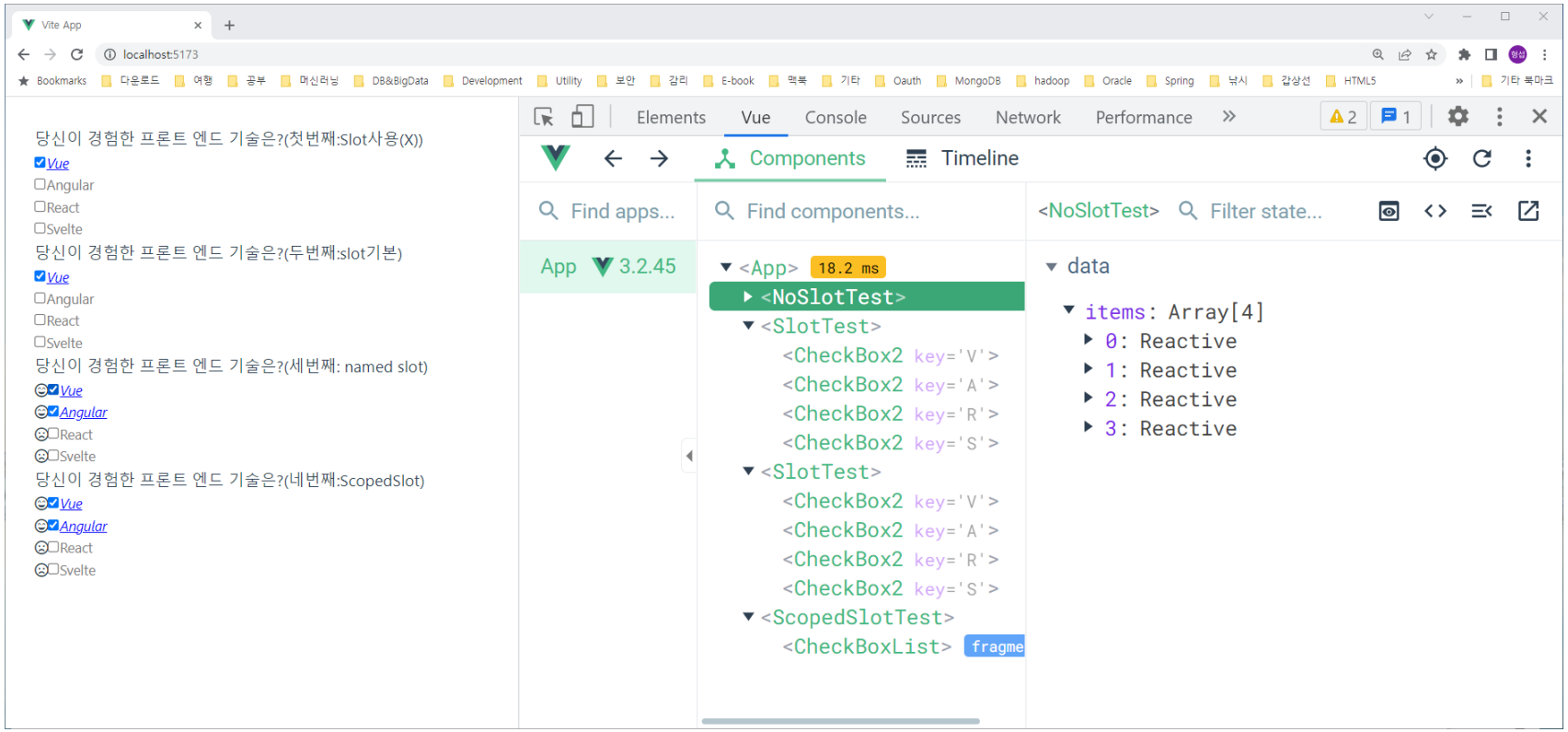
```vue
<template>
  <div>
    <h3>당신이 경험한 프론트 엔드 기술은?(네번째:ScopedSlot)</h3>
    <CheckBoxList :items="items" @check-changed="CheckBoxChanged">
      <template v-slot:icon="p1">
       <i v-if="p1.checked" class="far fa-grin-beam"></i>
       <i v-else class="far fa-frown"></i>
      </template>
      <template v-slot:default="p2">
       <span v-if="p2.checked" style="color:blue; text-decoration:underline;">
            <i>{{p2.label}}</i></span>
       <span v-else style="color:gray">{{p2.label}}</span>
      </template>
    </CheckBoxList>
  </div>
</template>

<script>
import CheckBoxList from './CheckBoxList.vue'

export default {
  name : "ScopedSlotTest",
  components : { CheckBoxList },
  data() {
    return {
      items : [
        { id:"V", checked:true, label:"Vue" },
        { id:"A", checked:false, label:"Angular" },
        { id:"R", checked:false, label:"React" },
        { id:"S", checked:false, label:"Svelte" },
      ]
    }
  },
  methods : {
    CheckBoxChanged(e) {
      let item = this.items.find((item)=> item.id === e.id);
      item.checked = e.checked;
    }
  }
}
</script>

<style>
@import url("https://cdnjs.cloudflare.com/ajax/libs/font-awesome/5.14.0/css/all.min.css");
</style>
```

2.4 범위 슬롯

❖ App.vue, main.js 변경 후 실행

❖ 실행 결과

- 화면은 기존 예제와 동일함



❖슬롯은 컴포넌트의 재사용성을 높여주는 유용한 기능

3. 동적 컴포넌트

❖ 동적 컴포넌트
- 동일한 위치에 여러 컴포넌트를 바인딩할 수 있는 기능을 제공함
- 〈componnent〉 요소와 v-bind 디렉티브를 이용함

❖ 프로젝트 생성과 초기화

```
npm init vue dynamic-component-test
cd dynamic-component-test
npm install
npm install bootstrap@5
```

- src/components , src/assets 디렉터리 내부의 모든 파일, 하위 디렉터리 삭제
- src/components 디렉터리에 다음 3개 파일 생성
  - LeyteGulfTab.vue
  - CoralSeaTab.vue
  - MidwayTab.vue

3. 동적 컴포넌트

- 예제 08-20 : src/components/CoralSeaTab.vue
  - 볼드체로 표현된 부분만 적절히 변경하여 MidwayTab.vue, LeyteGulfTab.vue도 작성

예제 08-20 : src/components/CoralSeaTab.vue

```
01: <template>
02:   <div class="tab">
03:       <h1>산호해 해전에 대해</h1>
04:       <div>{{ (new Date()).toTimeString() }}</div>
05:   </div>
06: </template>
07: <script>
08: export default {
09:     name : "CoralSeaTab"
10: }
11: </script>
<!-- MidwayTab.vue, LeyteGulfTab.vue 도 작성하세요-->
```

3. 동적 컴포넌트

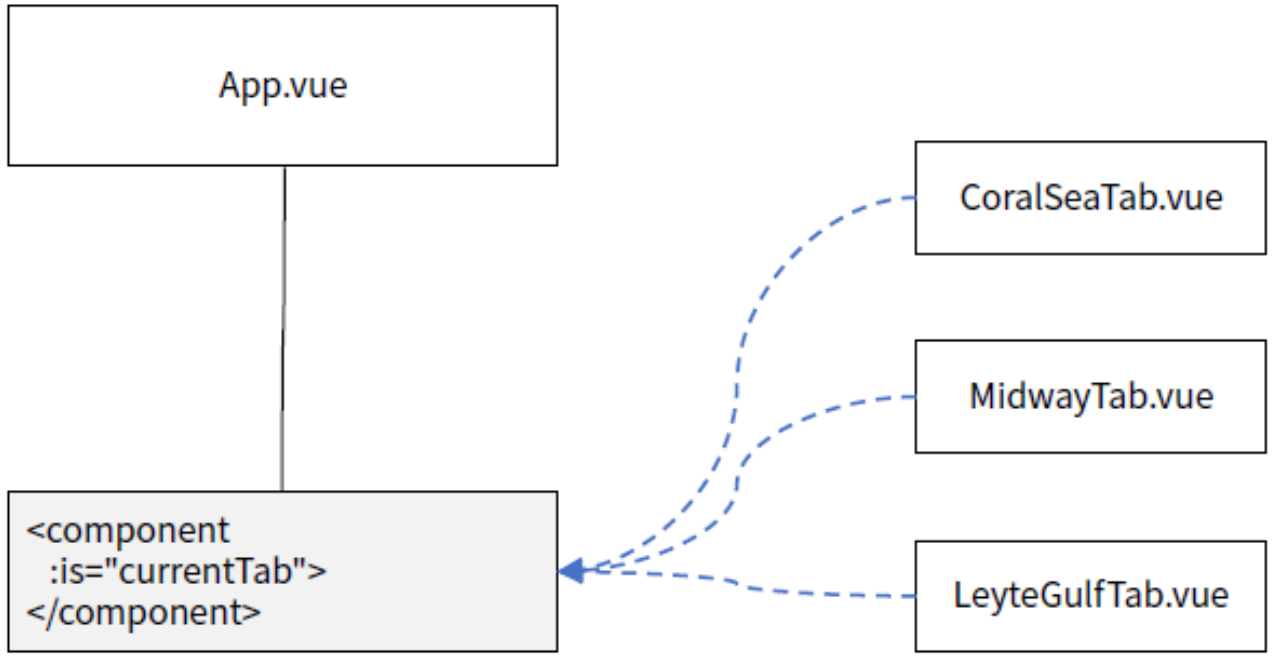
- 예제 08-21 : src/App.vue

```vue
<template>
  <div class="header">
    <h1 class="headerText">태평양 전쟁의 해전</h1>
    <nav>
      <ul class="nav nav-tabs nav-fill">
        <li v-for="tab in tabs" :key="tab.id" class="nav-item">
          <a style="cursor:pointer;" class="nav-link"
            :class="{ active : tab.id === currentTab }"
            @click="changeTab(tab.id)">{{tab.label}}</a>
        </li>
      </ul>
    </nav>
  </div>
  <div class="container">
    <component :is="currentTab"></component>
  </div>
</template>
<script>
import CoralSeaTab from './components/CoralSeaTab.vue'
import LeyteGulfTab from './components/LeyteGulfTab.vue'
import MidwayTab from './components/MidwayTab.vue'

export default {
  name: 'App',
  components : { CoralSeaTab, LeyteGulfTab, MidwayTab },
  data() {
    return {
      currentTab : 'CoralSeaTab',
      tabs : [
        { id:"CoralSeaTab", label:"산호해 해전" },
        { id:"MidwayTab", label:"미드웨이 해전" },
        { id:"LeyteGulfTab", label:"레이테만 해전" }
      ]
    }
  },
  methods : {
    changeTab(tab) {
      this.currentTab = tab;
    }
  }
}
</script>
<style>
.header { padding: 20px 0px 0px 20px; }
.headerText { padding: 0px 20px 40px 20px; }
.tab { padding: 30px }
</style>
```

3. 동적 컴포넌트

❖실행 결과

- src/main.js에 bootstrap을 import 하는 코드 추가후 실행



* is 특성에 바인한 currentTab 값이 등록한 컴포넌트 이름과 같을 때
<component />를 대신해 해당 컴포넌트가 렌더링됨.
* 등록한 컴포넌트 이름은 components 옵션으로 등록한 것을 의미함.

3. 동적 컴포넌트

❖주의사항

- 〈component〉의 is 특성에 바인딩하는 값은 등록한 컴포넌트의 이름만 허용
  - 컴포넌트 이름은 컴포넌트를 사용하기 위해 등록할 때의 이름
- 케밥 표기법, 파스칼 표기법, 카멜 표기법 모두 사용

```
export default {
  name: 'App',
  components : { CoralSeaTab, LeyteGulfTab, MidwayTab },
  ......
}
```

- 만일 다음과 같이 components 옵션을 등록했다면?
  - is 특성은 A, B, C 중 하나가 되어야 함.

```
{  "A" : CoralSeaTab, "B" : LeyteGulfTab, "C" : MidwayTab  }
```

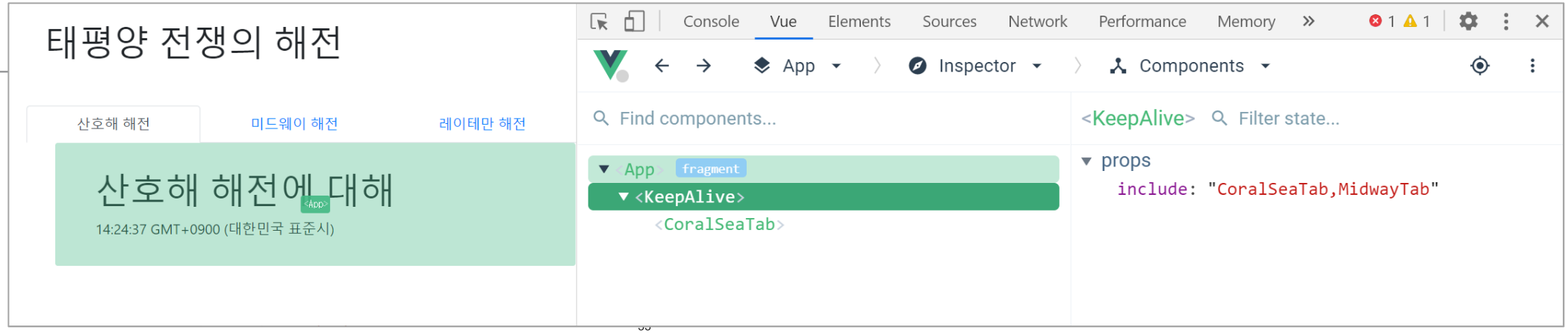
3. 동적 컴포넌트

❖ 이전까지의 실행 결과 화면에서 각 탭의 시간은 매번 바뀜
- 매번 실행되고 있음을 의미함

❖만일 정적 화면을 위해 캐싱여부를 지정하고 싶다면?
- 〈component〉를 〈keep-alive〉 태그로 감싸주고
- include, exclude 특성으로 지정함

예제 08-23 : src/App.vue에 〈keep-alive〉 적용

```
<!-- 예제 08-21의 14~16행을 다음과 같이 변경-->
<div class="container">
  <keep-alive include="CoralSeaTab,MidwayTab">
    <component :is="currentTab"></component>
  </keep-alive>
</div>
```



4. 컴포넌트에서의 v-model 디렉티브

❖ v-model 디렉티브
- 3장에서 학습한 내용
- 양방향 바인딩을 지원하기 위해 사용

❖ 컴포넌트에서의 v-model 디렉티브
- v-model은 컴포넌트에서 사용하면 미리 정의된 속성, 이벤트를 통해 양방향 바인딩처럼 부모 컴포넌트에서 사용할 수 있음

[부모 컴포넌트에서]

```
<child-component v-model:message="parentMessage" />
```

[자식 컴포넌트에서]

```
{
  name : "ChildComponent",
  props : { message : String },
  template : `<input type="text" :value="message"
      @input="$emit('update:message', $event.target.value)" />`
}
```

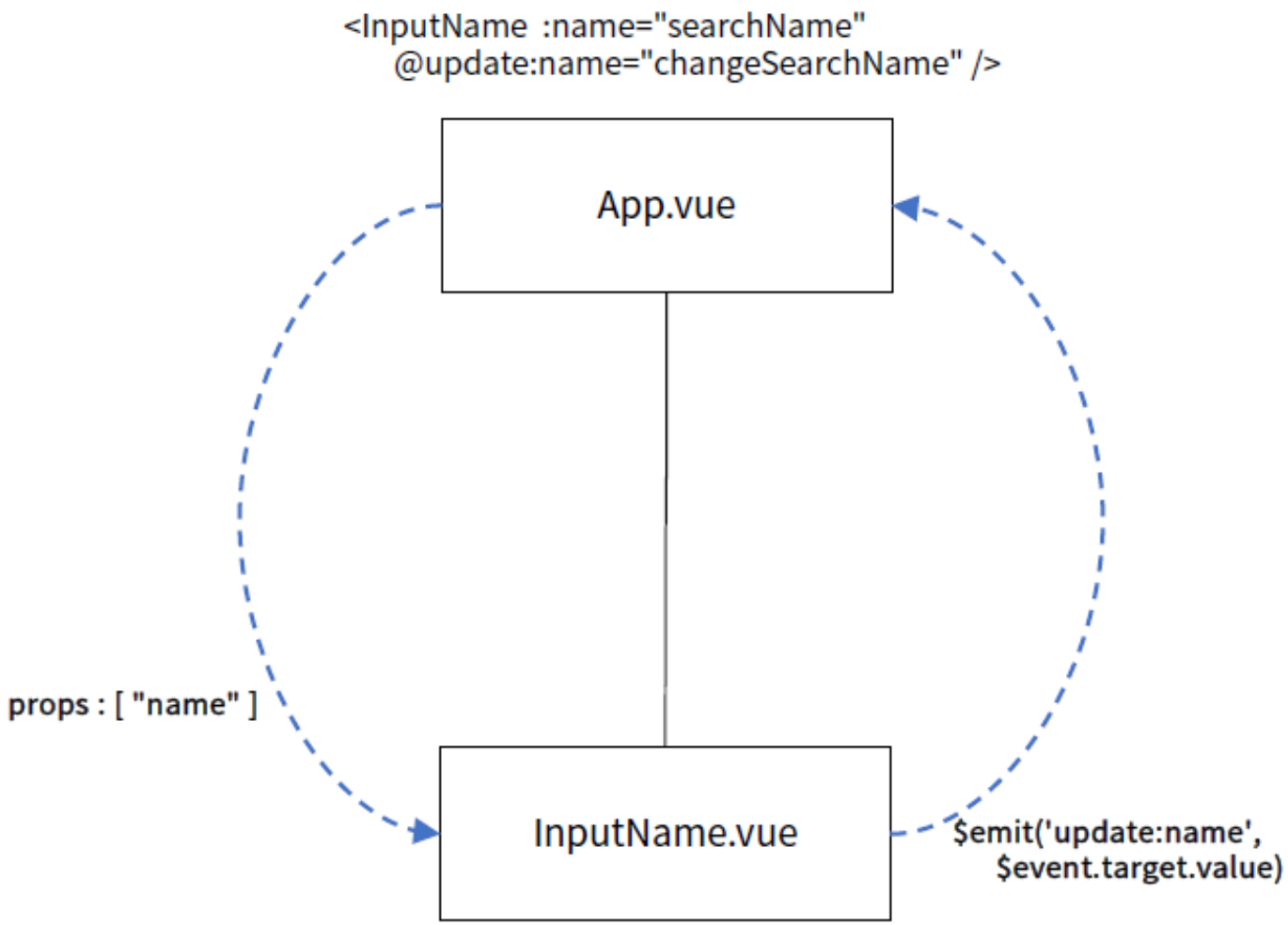
4. 컴포넌트에서의 v-model 디렉티브

❖ 프로젝트 생성

```
npm init vue v-model-test
cd v-model-test
npm install
```

- src/components 아래의 모든 파일, 하위 디렉터리 삭제

❖ 작성할 예제 구조



4. 컴포넌트에서의 v-model 디렉티브

❖ 예제 08-24~25 : src/components/InputName.vue, src/App.vue

```vue
<template>
    <input type="text"  :value="name"
      @input="$emit( 'update:name', $event.target.value )" />
</template>

<script>
export default {
    name : "InputName",
    props : ["name"],
}
</script>
```

- 예제 08-25는 v-model을 적용하기 전

```vue
<template>
  <div>
    <InputName :name="searchName"
      @update:name="changeSearchName" />
    <h3> 검색어 : {{searchName}}</h3>
  </div>
</template>

<script>
import InputName from './components/InputName.vue';

export default {
  name : "App",
  components : { InputName },
  data() {
    return { searchName : "John" }
  },
  methods : {
    changeSearchName(name) {
      console.log(name)
      this.searchName = name;
    }
  }
}
</script>
```

4. 컴포넌트에서의 v-model 디렉티브

❖ v-model 디렉티브 적용

예제 08-26 : src/App.vue 변경

```
//예제 08-25의 3행을 다음과 같이 변경합니다.
<template>
  <div>
    <InputName v-model:name="searchName" />
    <h3> 검색어 : {{searchName}}</h3>
  </div>
</template>
```

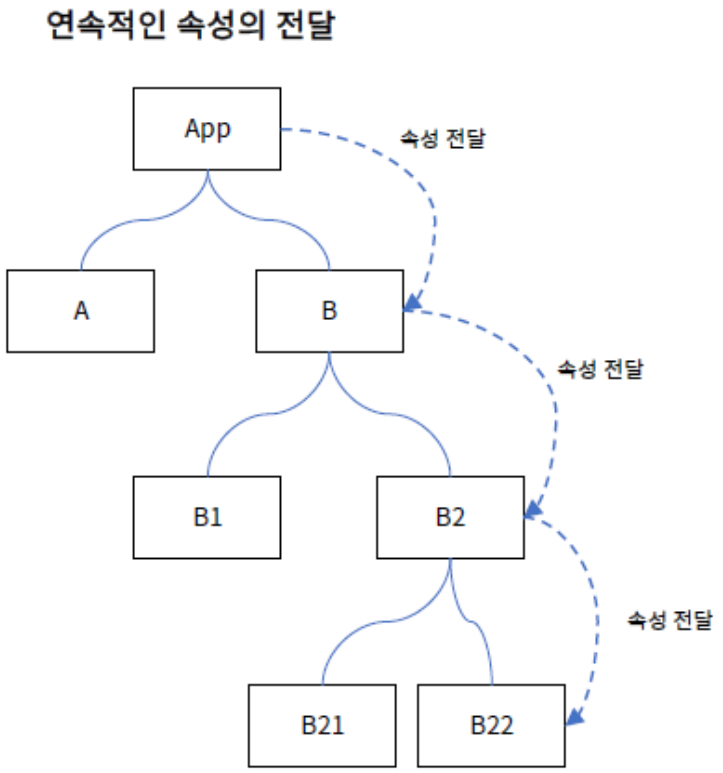
❖ 여러개의 v-model 디렉티브 지정

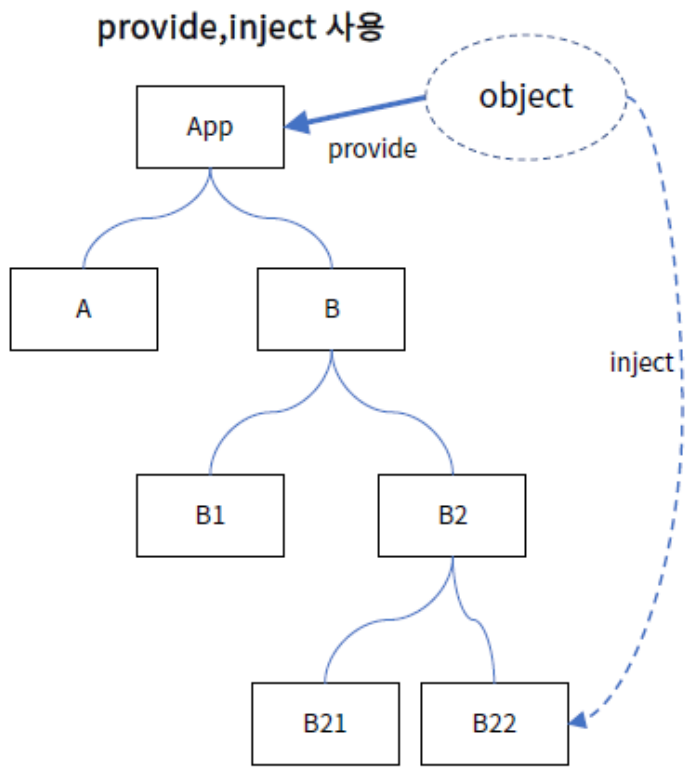
```
<InputUserInfo v-model:email="email"  v-model:mobile="mobile" />
```

5. provide, inject를 이용한 공용데이터 사용

❖ 속성(props)의 문제점
- Props Drilling의 문제
  - 컴포넌트의 계층 구조가 복잡해지면 계층 구조를 따라 연속적으로 속성을 전달해야 함

❖ provide/inject
- 이 방법을 이용하면 Props drilling을 해결할 수 있음
- 부모 컴포넌트에 공용 데이터를 제공
- 자식 컴포넌트에 필요한 데이터를 주입하여 이용





5. provide, inject를 이용한 공용데이터 사용

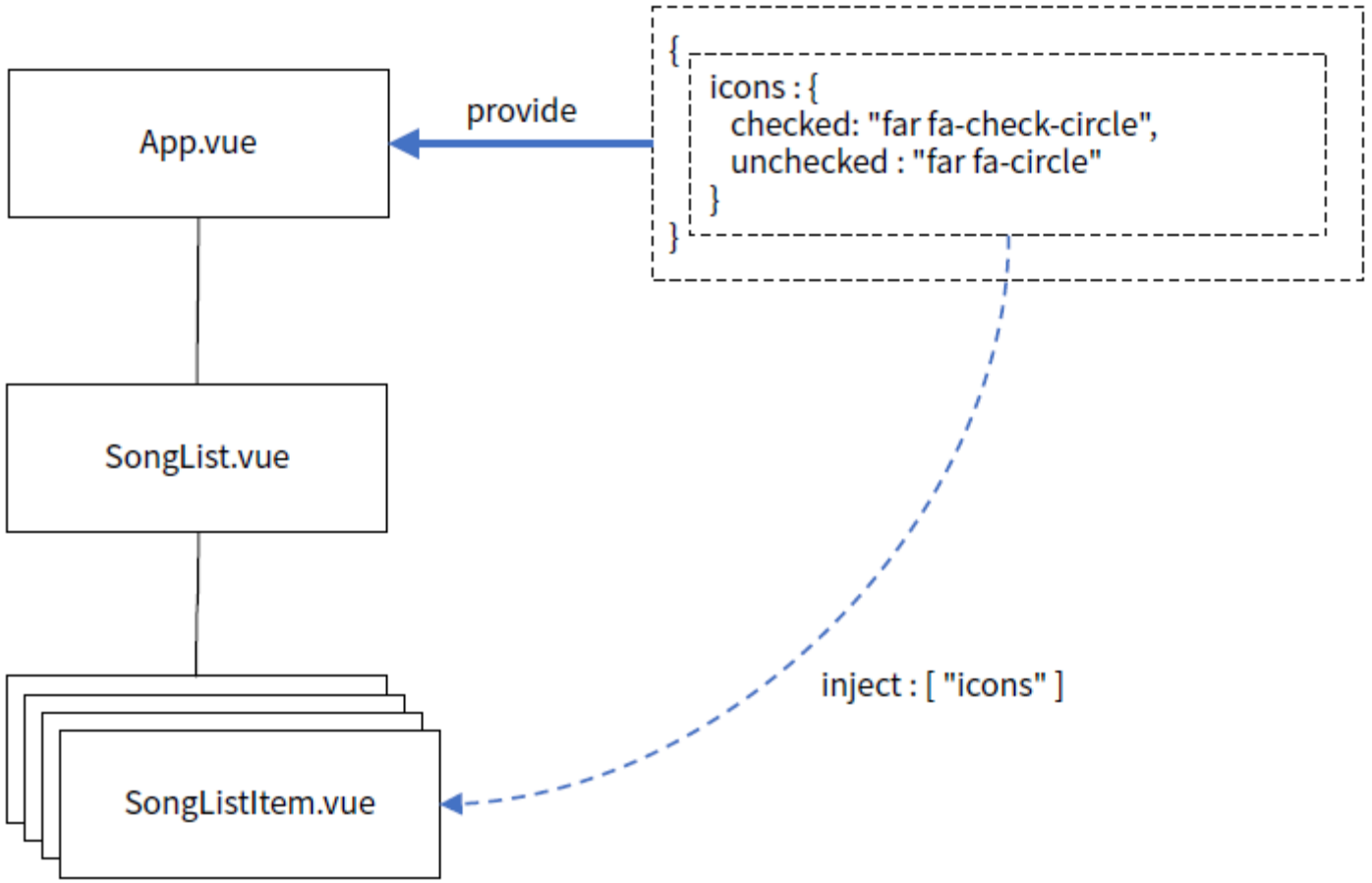
❖프로젝트 생성, 초기화

```
npm init vue provide-inject-teleport-test
cd provide-inject-teleport-test
npm install
```

- src/components 디렉터리 파일, 하위 폴더 삭제

❖작성할 컴포넌트

- App
- SongList
- SongListItem



5. provide, inject를 이용한 공용데이터 사용

❖ 예제 08-27 : src/App.vue

```vue
<template>
  <div>
    <h2>최신 인기곡</h2>
    <SongList :songs="songs" />
  </div>
</template>

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

export default {
  name : "App",
  components : { SongList },
  data() {
    return {
      songs : [
        { id:1, title:"Blueming", done:true },
        { id:2, title:"Dynamite", done:true },
        { id:3, title:"Lovesick Girls", done:false },
        { id:4, title:"마리아(Maria)", done:false },
      ]
    }
  },
  provide() {
    return {
      icons : {
        checked : "far fa-check-circle",
        unchecked : "far fa-circle",
      }
    }
  },
}
</script>
<style>
@import url("https://cdnjs.cloudflare.com/ajax/libs/font-awesome/5.14.0/css/all.min.css");
</style>
```

5. provide, inject를 이용한 공용데이터 사용

❖ 예제 08-28~29 : src/components/SongList.vue, SongListItem.vue

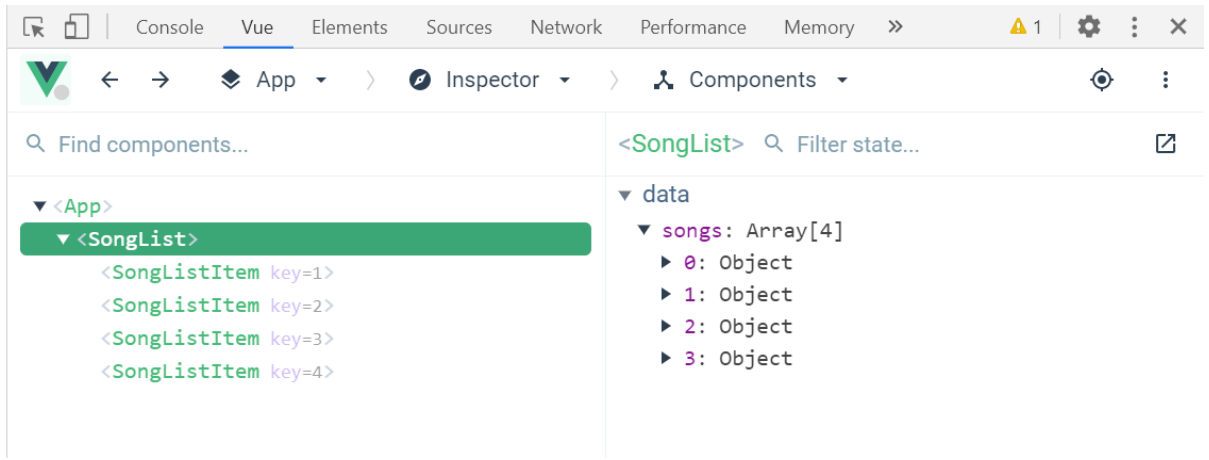
```vue
<template>
  <ul>
    <SongListItem v-for="s in songs" :key="s.id" :song="s" />
  </ul>
</template>

<script>
  import SongListItem from './SongListItem.vue'
  export default {
    name: "SongList",
    components: { SongListItem },
    props: ["songs"],
  }
</script>
```

```vue
<template>
  <li>
    <i :class="song.done ? icons.checked : icons.unchecked"></i> {{song.title}}
  </li>
</template>

<script>``
  export default {
    name: "SongListItem",
    inject: ["icons"],
    props: ["song"],
  }
</script>
```

최신 인기곡

- Blueming
- Dynamite
- Lovesick Girls
- 마리아(Maria)

Font Awesome 아이콘



5. provide, inject를 이용한 공용데이터 사용

❖provide로 제공되는 객체는 반응성을 제공하지 않음

- 변경하더라도 화면은 갱신되지 않음
- 반응성을 제공하려면 Vue 3의 Composition API가 지원하는 reactive, ref, computed 와 같은 API를 이용해야 함
  - 9장에서 학습할 내용

❖예제 08-30 : App.vue 변경

- computed 를 이용해 반응성을 가지도록 provide함

```
import { computed } from 'vue';

provide() {
  return {
    icons : {
      checked : "far fa-check-circle",
      unchecked : "far fa-circle",
    },
    doneCount : computed(()=> {
      return this.songs.filter((s)=>s.done === true).length
    })
  }
},
```

5. provide, inject를 이용한 공용데이터 사용

❖예제 08-31 : SongList.vue 변경

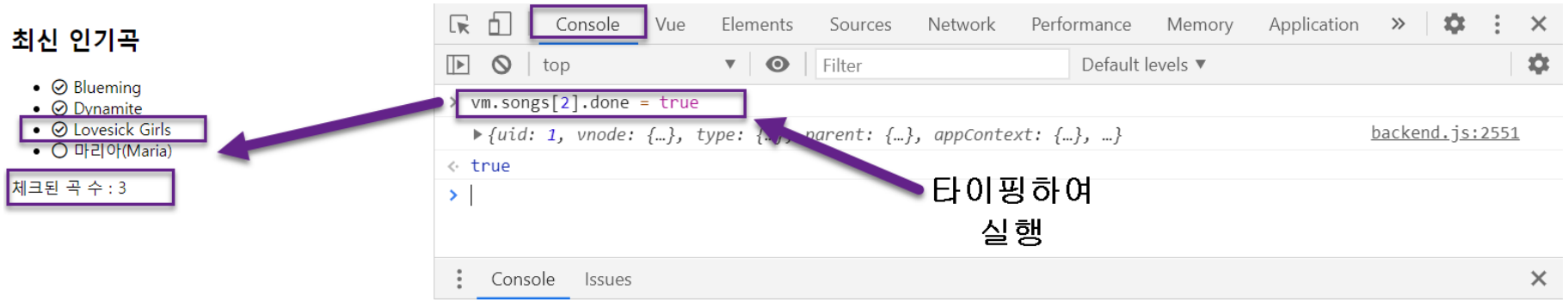
```
1   <template>
2     <ul>
3       <SongListItem v-for="s in songs" :key="s.id" :song="s" />
4     </ul>
5     <div>체크된 곡 수 : {{doneCount}}</div>
6   </template>
7
8   <script>
9     import SongListItem from './SongListItem.vue'
10    export default {
11      name: "SongList",
12      components: {
13        SongListItem
14      },
15      props: ["songs"],
16      inject : ["doneCount"],
17    }
18  </script>
```

5. provide, inject를 이용한 공용데이터 사용

❖예제 08-32 : src/main.js 변경

- 콘솔에서 데이터를 변경할 수 있도록 전역변수 vm에 Vue 인스턴스 할당

```
1  import { createApp } from 'vue'
2  import App from './App.vue'
3
4  import './assets/main.css'
5
6  const app = createApp(App);
7  app.config.unwrapInjectedRef = true
8  window.vm = app.mount('#app')
```



5. provide, inject를 이용한 공용데이터 사용

❖ 만일 Composition API가 제공하는 computed 를 사용하지 않았다면
- 반응성을 가지지 않음

[예제 08-30의 13~15행을 아래와 같이 변경]

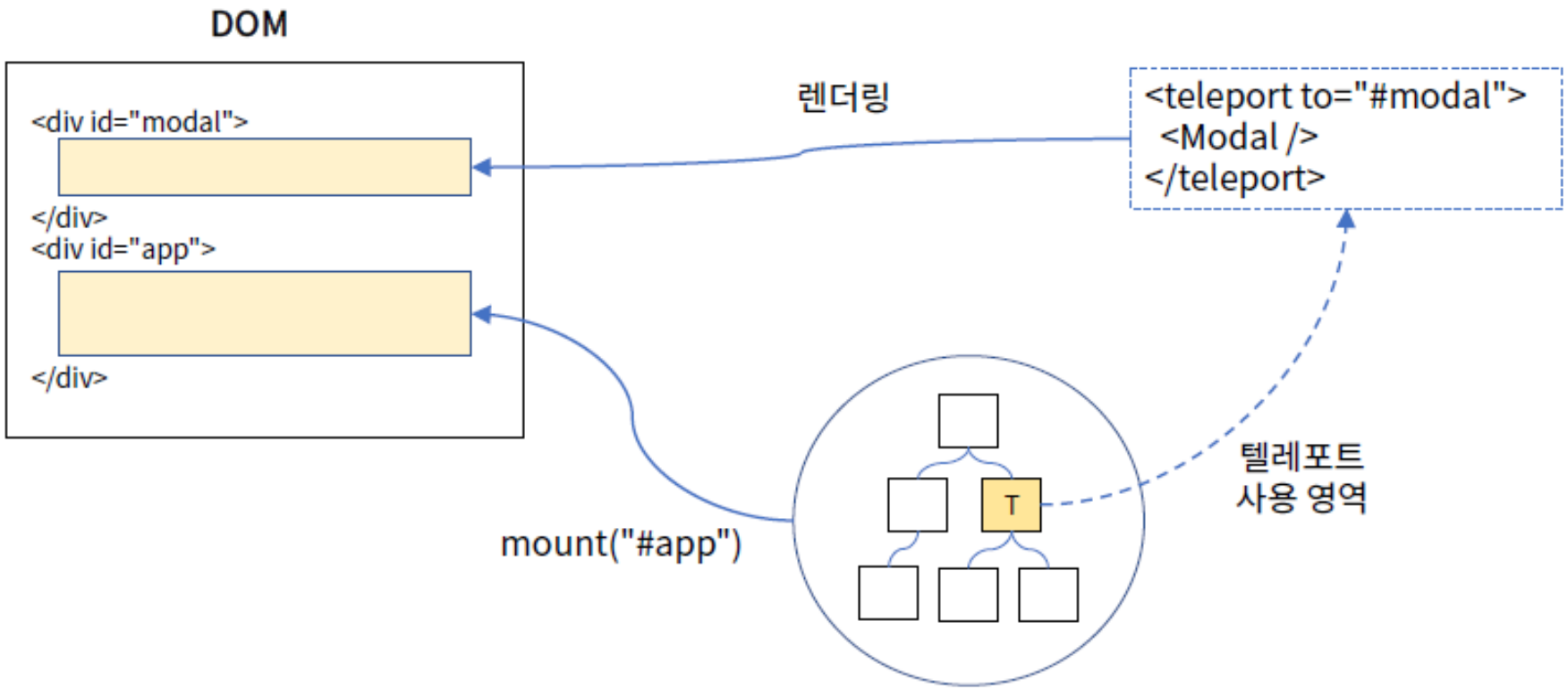
```
//import { computed } from 'vue'  --> 이 줄은 주석 처리하세요.
doneCount : this.songs.filter((s)=>s.done === true).length
```

❖ provide/inject 는 공용 데이터를 읽기 전용으로 이용할 때 사용
- 반응성을 제공하는 것은 추천되지 않음
- 반응성을 제공한 후 하위 컴포넌트 중 하나에서 데이터를 자주 변경하면?
  - 어디서 변경했는지를 추적하기 힘들어짐
- 반응성을 제공해야 한다면 Vuex, Pinia 사용을 권장함

6. 텔레포트

❖텔레포트

- 애플리케이션에서 모달, 툴팁과 같이 메인 화면과 독립적이면서 공유 UI를 제공
- 기본 렌더링 위치( )가 아닌 다른 요소에 렌더링함



6. 텔레포트

❖예제 08-33 : src/components/Modal.vue

```
1   <template>
2     <div class="modal">
3       <div class="box">
4         처리중
5       </div>
6     </div>
7   </template>
8
9   <script>
10  export default {
11    name : "Modal"
12  }
13  </script>
14
15  <style scoped>
16  .modal { display: block; position: fixed; z-index: 1; ···
20  .box {   display: flex; flex-direction: column; ···
25  </style>
```

6. 텔레포트

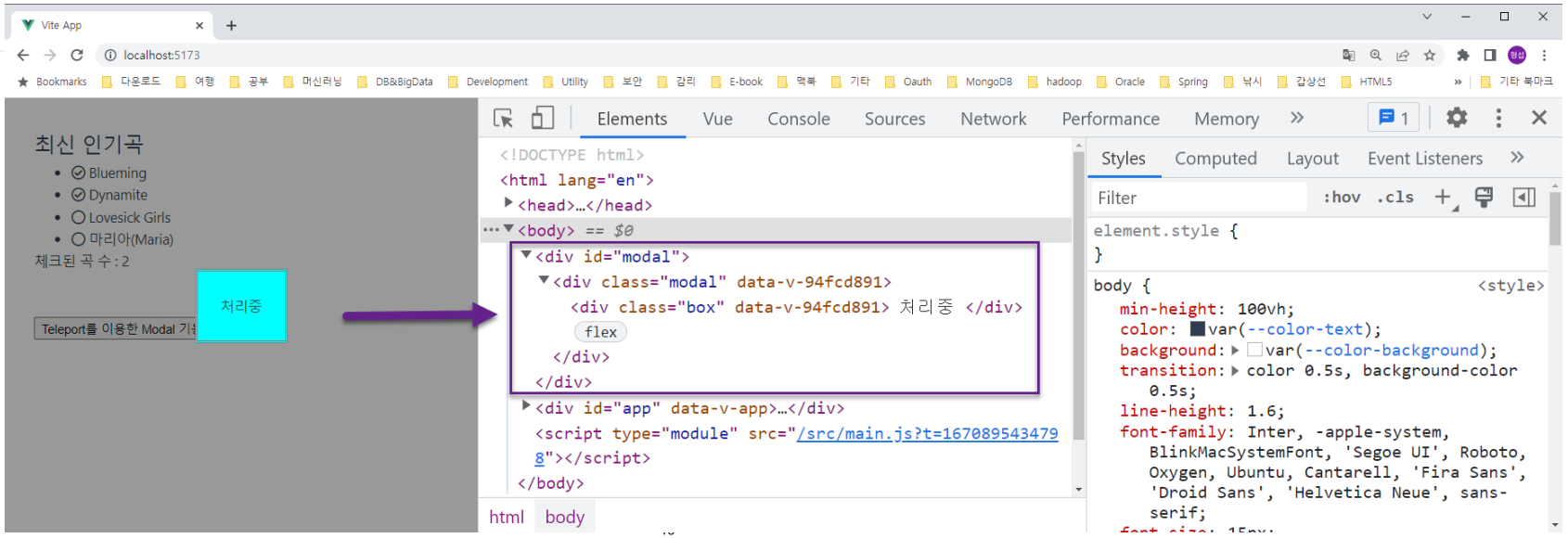
❖ 예제 08-34 : src/App.vue 변경

```vue
<template>
  <div>
    <h2>최신 인기곡</h2>
    <SongList :songs="songs" />
    <br /><br />
    <button @click="changeModal">Teleport를 이용한 Modal 기능</button>
    <teleport to="#modal">
      <Modal v-if="isModal" />
    </teleport>
  </div>
</template>

<script>
import { computed } from 'vue';
import SongList from './components/SongList.vue'
import Modal from './components/Modal.vue'

export default {
  name : "App",
  components : { SongList, Modal },
  data() {
    return {
      songs : [ ···
      ],
      isModal : false,
    }
  },
  methods : {
    changeModal() {
      this.isModal = true;
      setTimeout(()=>{ this.isModal = false }, 2000);
    }
  },
  provide() { ···
  },
}
</script>
```

6. 텔레포트

❖ 예제 08-35 : index.html 변경

```
1  <!DOCTYPE html>
2  <html lang="en">
3    <head>
4      <meta charset="UTF-8">
5      <link rel="icon" href="/favicon.ico">
6      <meta name="viewport" content="width=device-width, initial-scale=1.0">
7      <title>Vite App</title>
8    </head>
9    <body>
10     <div id="modal"></div>
11     <div id="app"></div>
12     <script type="module" src="/src/main.js"></script>
13   </body>
14 </html>
```



7. 비동기 컴포넌트

❖지금까지의 컴포넌트는...

- index.html을 로드하고 관련된 .js 파일을 모두 로드한 후 마운트되고 렌더링됨
  - 화면에서 당장 그 컴포넌트가 필요하지 않아도 미리 로딩함
  - 초기 로딩 시간 지연
- 이런 경우에 비동기 컴포넌트를 사용할 수 있음
  - 컴포넌트가 사용되는 시점에 .js 파일을 로딩함
  - 이를 위해 빌드할 때 비동기 컴포넌트를 위한 .js 파일을 분리하여 번들링함

❖비동기 컴포넌트 사용법

[일반적인 컴포넌트의 임포트]

```
import SyncComponent from './SyncComponent.vue';
```

[비동기 컴포넌트의 임포트]

```
import { defineAsyncComponent } from 'vue'

const AsyncComponent = defineAsyncComponent(
  () => import('./AsyncComponent.vue')
)
```

7. 비동기 컴포넌트

❖ 기존 dynamic-component-test 프로젝트 예제 작성

- 패키지 추가 : npm install p-min-delay

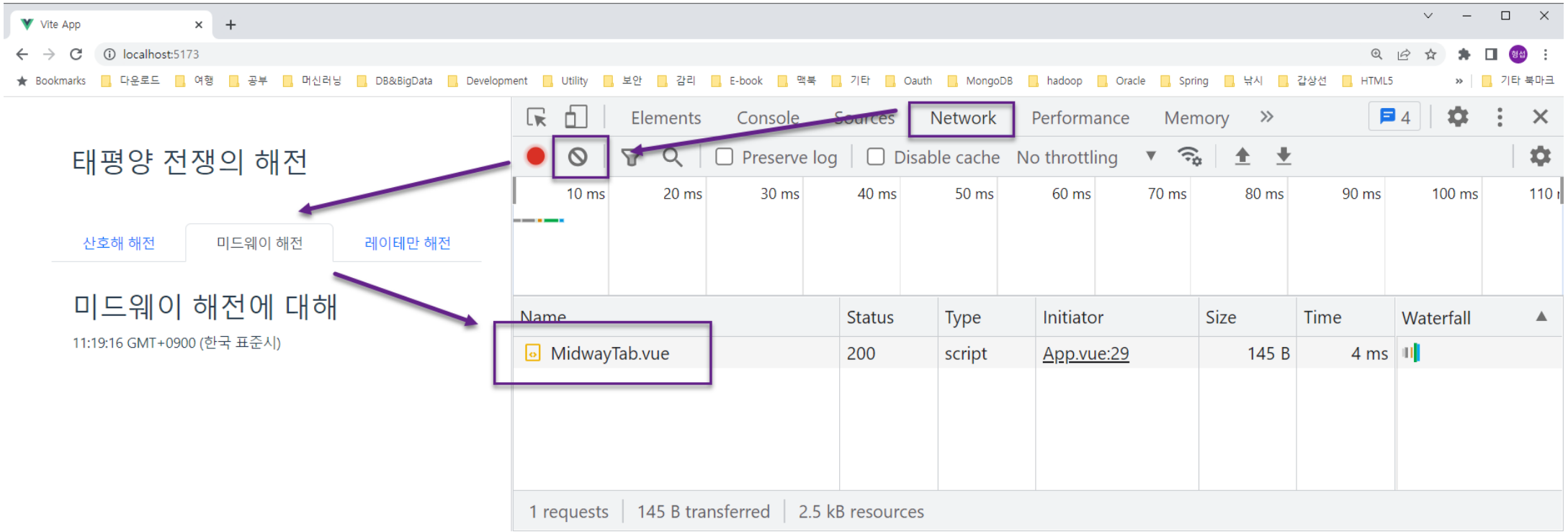
❖예제 08-36 : src/App.vue 변경

```
05:     <!-- <keep-alive>는 제거합니다.-->
06:     <component :is="currentTab"></component>
07:     <!-- </keep-alive>는 제거합니다.-->

11: <script>
12: import { defineAsyncComponent } from 'vue'
13: import pMinDelay from 'p-min-delay'
14:
15: const CoralSeaTab = defineAsyncComponent(
16:   () => pMinDelay(import('./components/CoralSeaTab.vue'), 2000)
17: )
18: const LeyteGulfTab = defineAsyncComponent(
19:   () => pMinDelay(import('./components/LeyteGulfTab.vue'), 2000)
20: )
21: const MidwayTab = defineAsyncComponent(
22:   () => pMinDelay(import('./components/MidwayTab.vue'), 2000)
23: )
```

7. 비동기 컴포넌트

❖실행 결과 확인

- 2초의 지연시간과 함께 컴포넌트 로딩
- 한번 로딩되면 다시 로딩되지 않음



7. 비동기 컴포넌트

❖ defineAsyncComponent() 함수의 옵션

```
const AsyncComp = defineAsyncComponent({
  //비동기로 로딩할 대상 컴포넌트
  loader : import('./AsyncComponent.vue'),
  //비동기 컴포넌트를 로딩하는 지연시간동안 보여줄 컴포넌트
  loadingComponent : LoadingComponent,
  //비동기 컴포넌트의 로딩 실패시에 보여줄 컴포넌트
  errorComponent : ErrorComponent,
  //로딩 컴포넌트를 보여주기전 지연 시간
  delay : 200,
  //로딩 제한 시간. 이 시간을 넘으면 errorComponent가 화면에 나타남
  timeout : 5000,
  //컴포넌트가 suspensible 한지 여부를 지정함. 기본값 true
  suspensible : true,
  //비동기 컴포넌트 로딩 실패시 실행할 함수
  // error : 에러메시지 객체, retry : 재시도를 수행할 함수
  // fail : 실패 처리를 할 함수, attempts : 재시도 횟수
  onError(error, retry, fail, attempts) {
    if (error.message.match(/fetch/) && attempts <= 3) {
      retry()
    } else {
      fail()
    }
  }
})
```

7. 비동기 컴포넌트

❖지연시간동안 Spinner UI 보여주기

- npm install --save vue-csspin

❖예제 08-37 : src/components/Loading.vue

```
1   <template>
2     <VueCsspin message="Loading" spin-style="cp-flip" />
3   </template>
4
5   <script>
6   import { VueCsspin } from 'vue-csspin'
7   import 'vue-csspin/dist/vue-csspin.css'
8
9   export default {
10    name : "Loading",
11    components : { VueCsspin },
12  }
13  </script>
```

7. 비동기 컴포넌트

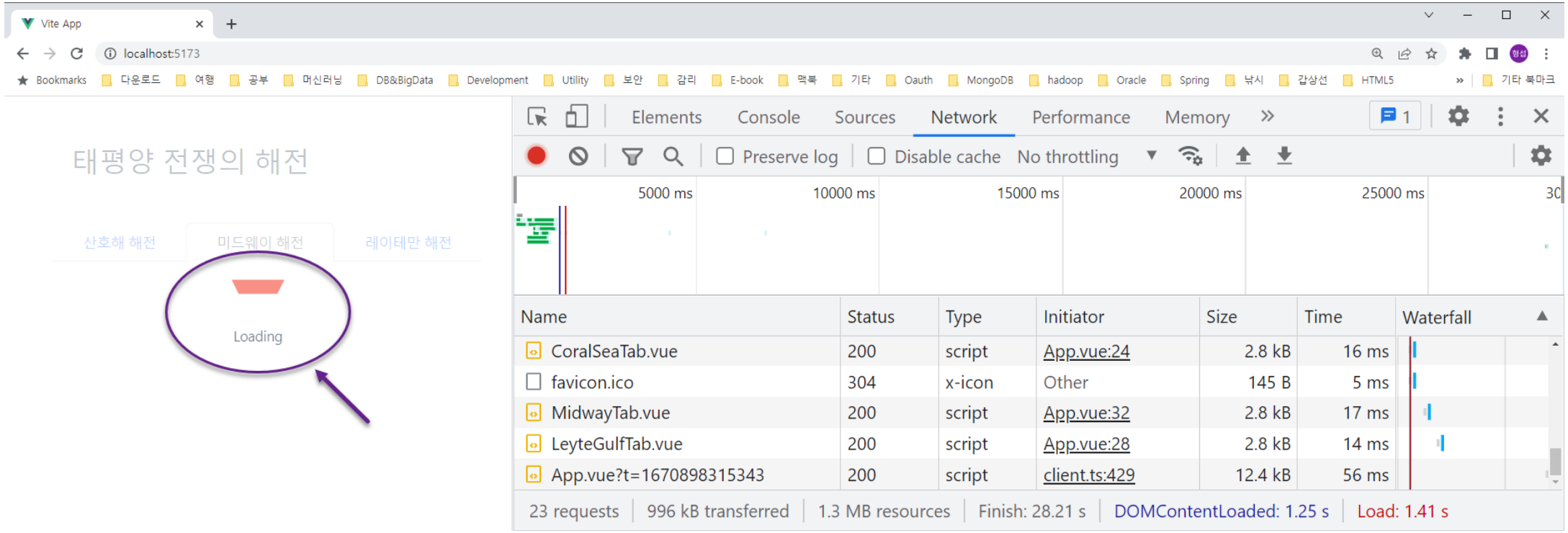
❖예제 08-38 : src/App.vue 변경

예제 08-38 : src/App.vue 변경

```
import { defineAsyncComponent } from 'vue'
import pMinDelay from 'p-min-delay'
import Loading from './components/Loading.vue';

const CoralSeaTab = defineAsyncComponent({
  loader : () => pMinDelay(import('./components/CoralSeaTab.vue'), 2000),
  loadingComponent : Loading,
})
const LeyteGulfTab = defineAsyncComponent({
  loader : () => pMinDelay(import('./components/LeyteGulfTab.vue'),2000),
  loadingComponent : Loading,
})
const MidwayTab = defineAsyncComponent({
  loader : () => pMinDelay(import('./components/MidwayTab.vue'), 2000),
  loadingComponent : Loading,
})
```

7. 비동기 컴포넌트

❖예제 실행 결과



8. 마무리

지금까지 학습한 컴포넌트에 대한 심화된 내용은 모두 컴포넌트의 재사용성을 높이기 위한 방법입니다. 분량이 많았지만 꼼꼼하게 학습해야 할 내용들입니다.

가장 먼저 학습한 내용은 단일 파일 컴포넌트에서 CSS 스타일의 충돌을 피하는 방법입니다. 범위 CSS, CSS 모듈을 이용해 CSS 클래스명의 이름 충돌을 피하는 방법을 살펴보았습니다.

다음으로 학습한 내용은 슬롯입니다. 슬롯은 부모 컴포넌트에게서 자식 컴포넌트로 템플릿을 전달할 수 있도록 기능을 제공합니다. 또한 명명된 슬롯을 이용하면 여러 개의 슬롯을 사용할 수 있습니다. 범위 슬롯은 부모 컴포넌트에서 템플릿에 바인딩되는 값을 자식 컴포넌트가 제공할 수 있도록 합니다.

동적 컴포넌트는 한 위치에 여러 컴포넌트 중 하나의 컴포넌트를 렌더링하도록 할 수 있습니다. is 특성 값을 컴포넌트의 이름으로 지정하여 제어합니다.

provide/inject 기능은 애플리케이션의 읽기 전용의 공용 데이터를 사용하고자 할 때 적합합니다. 부모 컴포넌트에서 데이터를 제공(provide)하고, 하위 트리의 자식 컴포넌트에서는 필요한 데이터를 주입(inject)하여 사용합니다.

비동기 컴포넌트는 화면에 필요한 컴포넌트를 필요한 때에 로딩하도록 하여 초기화면의 로딩 속도를 빠르게 할 수 있습니다.