KB금융그룹 | 국민의 평생 금융파트너

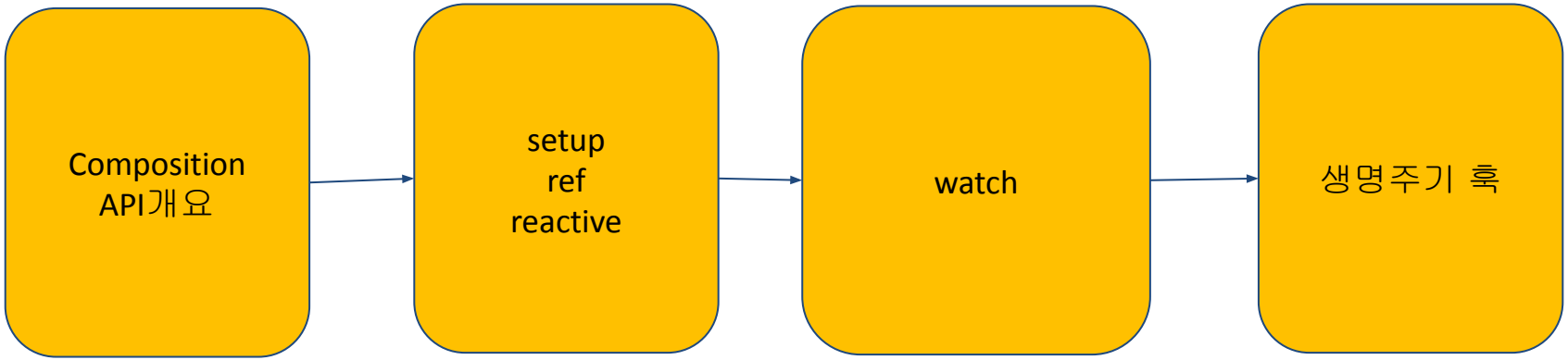
2024년 상반기 K-디지털 트레이닝

Composition API

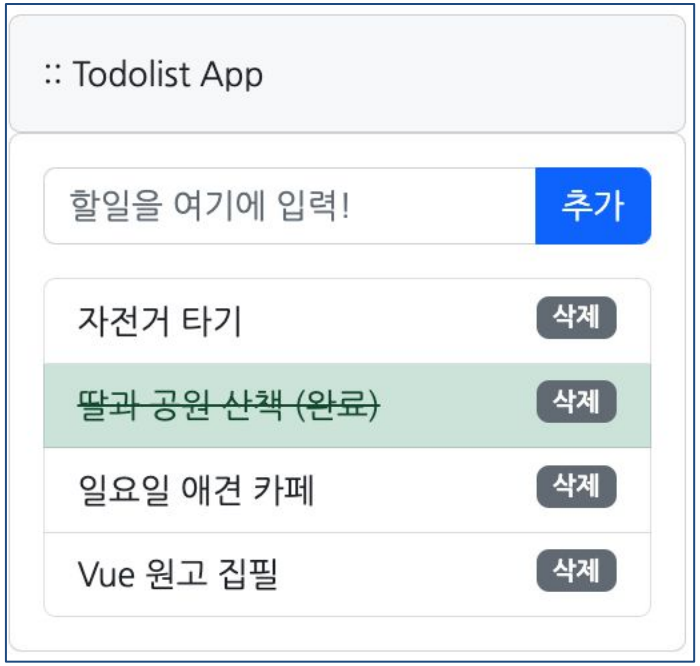
[KB] IT's Your Life

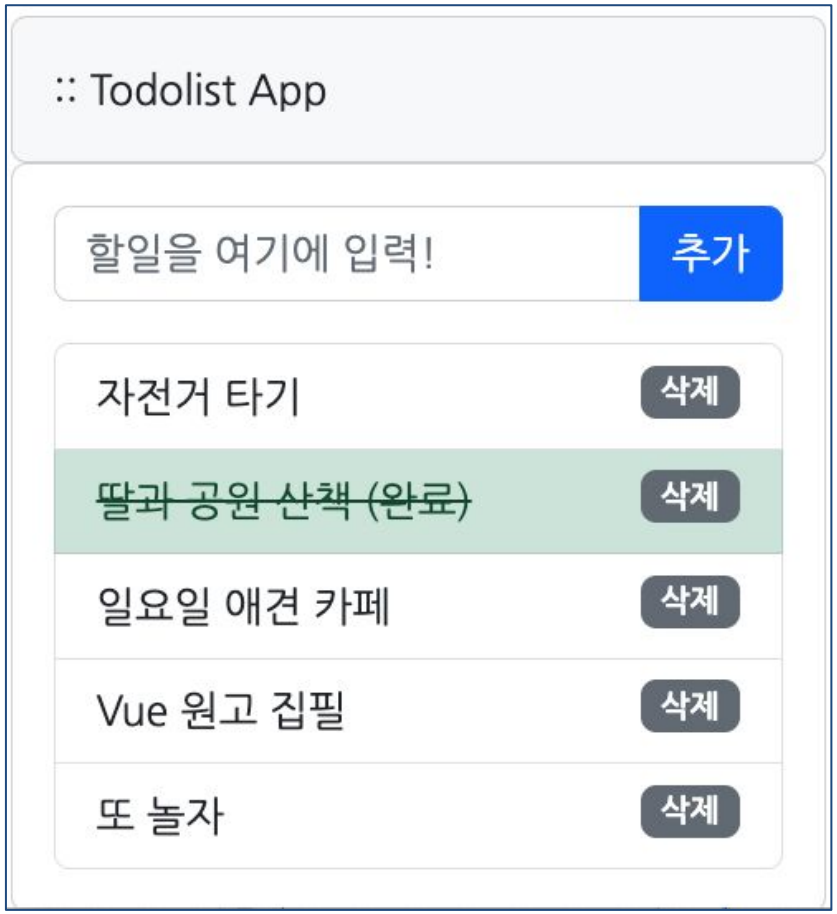
KB 국민은행

커리큘럼



오늘의 목표


:: Todolist App
할일을 여기에 입력!
추가
자전거 타기
삭제
딸과 공원 산책 (완료)
삭제
일요일 애견 카페
삭제
Vue 원고 집필
삭제


:: Todolist App
할일을 여기에 입력!
추가
자전거 타기
삭제
딸과 공원 산책 (완료)
삭제
일요일 애견 카페
삭제
Vue 원고 집필
삭제
또 놀자
삭제

Composition API

Composition API 개요

- 지금까지는 **Vue2**에서 사용하던 **Option API**를 사용하였음.
- **Vue3**에서 추가
- **Vue**객체 인스턴스 생성시 컴포넌트에 대한 정의를 좀더 명확하게 명시하고 구조적으로 기술
- 재사용성을 높이는 기술 방식
- 기존의 **Vue2**의 **Option API**방식은 **data, methods, computed**가 컴포넌트별로 분리되지 않고 섞여서 기술됨.
  - 컴포넌트 로직 재사용이 용이하지 않음.
  - mixin도 재사용성을 위해 컴포넌트별로 확인하기 불편

```
<script>
export default {
    name : "OptionAPI",
    data(){
        return {
            name : "",
            x: 0,
            y: 0
        }
    }
    computed : {
        result() {
            return parseInt(this.x, 10) + parseInt(this.y, 10)
        }
    },
    mounted() {
        this.name = "john",
        this.x = 10,
        this.y = 20;
    },
    methods : {
        changeX(strX) {
            let x = parseInt(strX, 10);
            this.x = isNaN(x) ? 0 : x;
        },
        changeY(strY) {
            let y = parseInt(strY, 10);
            this.y = isNaN(y) ? 0 : y;
        },
        changeName(name) {
            this.name = name.trim().length < 2 ? "" : name.trim();
        }
    }
}
```

cal

name

Composition API 개요

- **Composition API와 Option API비교**

```
<script>
import { reactive, computed, onMounted } from 'vue';

export default {
    name : "CompositionAPI",
    setup() {
        const nameData = reactive({name : ""})
        const changeName = (name) => {
            console.log(name);
            nameData.name = name.trim().length < 2 ? "" : name.trim();
        }
        onMounted(() => nameData.name = "john");

        const calcData = reactive({ x: 0, y: 0});
        const result = computed(() => parseInt(calcData.x, 10) + parseInt(calcData.y, 10))
        onMounted(() => {
            calcData.x = 10;
            calcData.y = 20;
        })
        const changeX = (strX) => {
            let x = parseInt(strX, 10);
            calcData.x = isNaN(x) ? 0 : x;
        }
        const changeY = (strY) => {
            let y = parseInt(strY, 10);
            calcData.y = isNaN(y) ? 0 : y;
        }
    }
}
```

name

cal

Composition API 개요

- **Composition API**는 재사용 가능한 함수로 분리도 가능

```
<script>
import { reactive, computed, onMounted } from 'vue';

const useName = (name = 'john') => {
    const nameData = reactive({name : ""})
    const changeName = (name) => {
            console.log(name);
            nameData.name = name.trim().length < 2 ? "" : name.trim();
    }
    onMounted(() => nameData.name = "john");
    return { nameData, changeName };
}

const useCalc = ( x= 0, y = 0) => {
    const calcData = reactive({ x: 0, y: 0});
    const result = computed(
                () => parseInt(calcData.x, 10) + parseInt(calcData.y, 10)
    );
    onMounted(() => {
        calcData.x = 10;
        calcData.y = 20;
    })
    const changeX = (strX) => {
        let x = parseInt(strX, 10);
        calcData.x = isNaN(x) ? 0 : x;
    }
    const changeY = (strY) => {
        let y = parseInt(strY, 10);
        calcData.y = isNaN(y) ? 0 : y;
    }
    return {calcData, result, changeX, changeY };
}
```

name

cal

```
export default {
    name : "CompositionAPI",
    setup() {
        const nameObj = useName("smith");
        const calcObj = useCalc(100, 200);

        return { ...nameObj,  ...calcObj}
    }
}
```

setup()

setup

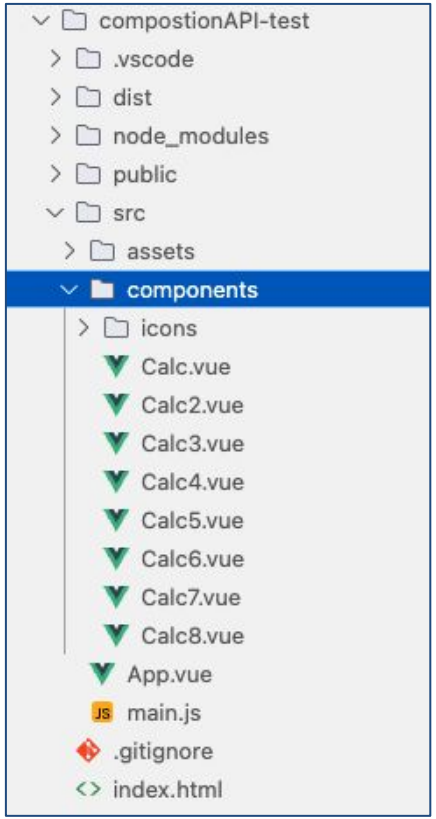
- **setup()**
  - Option API의 data, methods, computed 초기화 작업을 담당
  - 컴포넌트 상태 초기화
  - 생명주기 주요 메서드 정의
  - 기타 메서드 등록
  - 리턴 속성 값은 템플릿에서 사용 가능

```
<template>
    <div>
        X : <input type="text" v-model.number="x" /><br/>
        Y : <input type="text" v-model.number="y" /><br/>
    </div>
</template>

<script>
import { ref } from 'vue';

export default {
    name:"Calc",
    setup () {
        const x = ref(10);
        const y = ref(20);
        return {x,y}
    }
}
</script>
```

setup

- **setup() 함수**

```
administrator@MacBook-Pro myNode % npm init vue compositionAPI-test

Vue.js - The Progressive JavaScript Framework

✔ Package name: … compositionapi-test
✔ Add TypeScript? … No / Yes
✔ Add JSX Support? … No / Yes
✔ Add Vue Router for Single Page Application development? … No / Yes
✔ Add Pinia for state management? … No / Yes
✔ Add Vitest for Unit Testing? … No / Yes
✔ Add an End-to-End Testing Solution? › No
✔ Add ESLint for code quality? … No / Yes
✔ Add Vue DevTools 7 extension for debugging? (experimental) … No / Yes

Scaffolding project in /Users/administrator/Documents/myNode/compositionAPI-test...

Done. Now run:

  cd compositionAPI-test
  npm install
  npm run dev
```

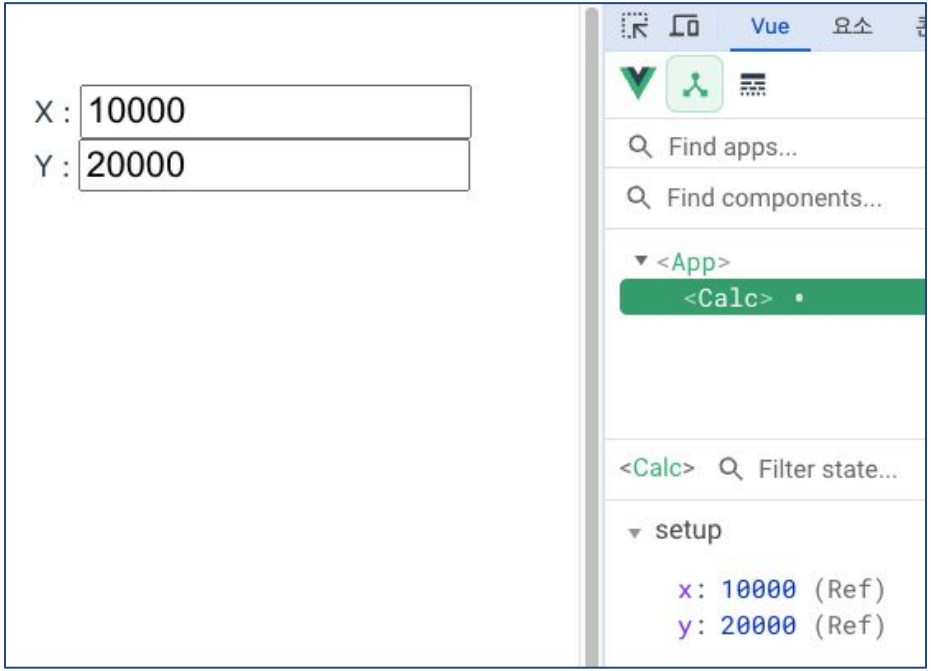


setup

- **setup() 함수**
- **Option API와 import하여 사용하는 방법은 동일**

App.vue

```vue
<template>
  <div>
    <Calc />
  </div>
</template>

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

export default {
  name:"App",
  components : { Calc },
}
</script>
```



setup

- **setup() 함수**
- **반응성 데이터 함수**
  - ref()
  - reactive()
- **ref**
  - 기본형 값의 반응성을 가진 참조형 데이터 생성시 사용
  - ref(a = 100), 파라메터의 초기값 설정
  - setup() 함수 리턴값으로 설정 → 템플릿에서 사용
  - setup() 함수 내에서도 사용 가능, 다른 함수에서 참조하여 사용 가능
  - Option API에서도 this로 참조하여 사용할 수 있음. value속성 사용. 권장하지는 않음.

```
<template>
    <div>
        X : <input type="text" v-model.number="x" /><br/>
        Y : <input type="text" v-model.number="y" /><br/>
        <button @click="calcAdd">계산</button><br />
        <div>결과 : {{result}}</div>
    </div>
</template>

<script>
import { ref } from 'vue';

export default {
    name:"Calc2",
    setup () {
        const x = ref(10);
        const y = ref(20);
        const result = ref(30);
        const calcAdd = () => {
            result.value = x.value + y.value;
        }

        return { x, y, result, calcAdd };
    }
}
</script>
```

setup

- **setup() 함수**
- **반응성 데이터 함수**
  - ref()
  - reactive()
- **reactive()**
  - 객체에 대한 반응성 설정
  - 리턴한 객체에 대해 반응성 적용됨.
  - 객체의 일부속성을 리턴한 경우 반응성 적용 불가
  - 반응성 적용되지 않으면 데이터가 변경되어도 화면에 있는 값이 변경되지 않음.

```
<template>
    <div>
        X : <input type="text" v-model.number="state.x" /><br/>
        Y : <input type="text" v-model.number="state.y" /><br/>
        <div>결과 : {{result}}</div>
    </div>
</template>

<script>
import { reactive, computed } from 'vue';

export default {
    name:"Calc4",
    setup () {
        const state = reactive({ x:10, y:20 });
        const result = computed(()=>{
            return state.x + state.y;
        })
        return { state, result }
    }
}
</script>
```

return {state.x} 하면 반응성 제거됨.

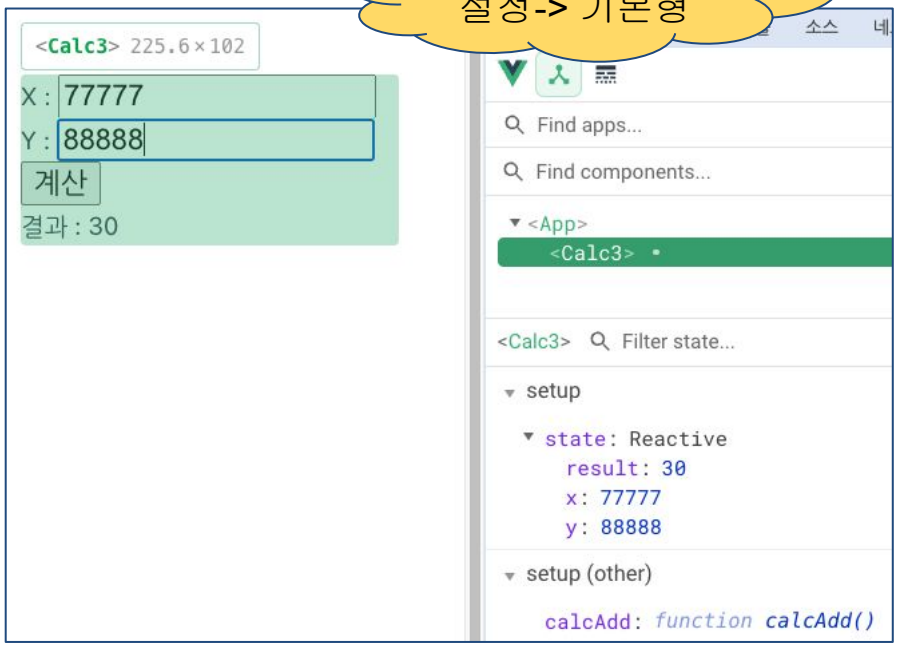
```
export default {
    name:"Calc4",
    setup () {
        const state = reactive({ x:10, y:20 });
        const result = computed(()=>{
            return state.x + state.y;
        })
        console.log('computed에서 생성된 계산 결과값>>> ', result.value);
```

computed()에서 계산된 결과를 js에서 접근시 result.value해주어야함.

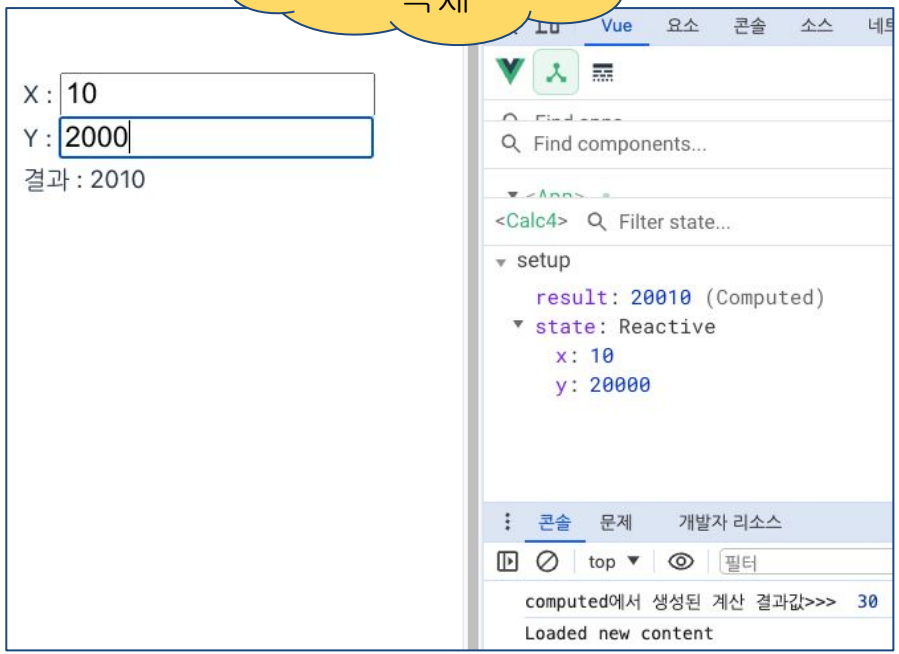
setup

- **setup() 함수**
- **ref()와 reactive()**

ref() 반응성 설정-> 기본형



reactive() 반응성 설정 -> 객체



watch

watch

- **Option API의 watch옵션과 동일**

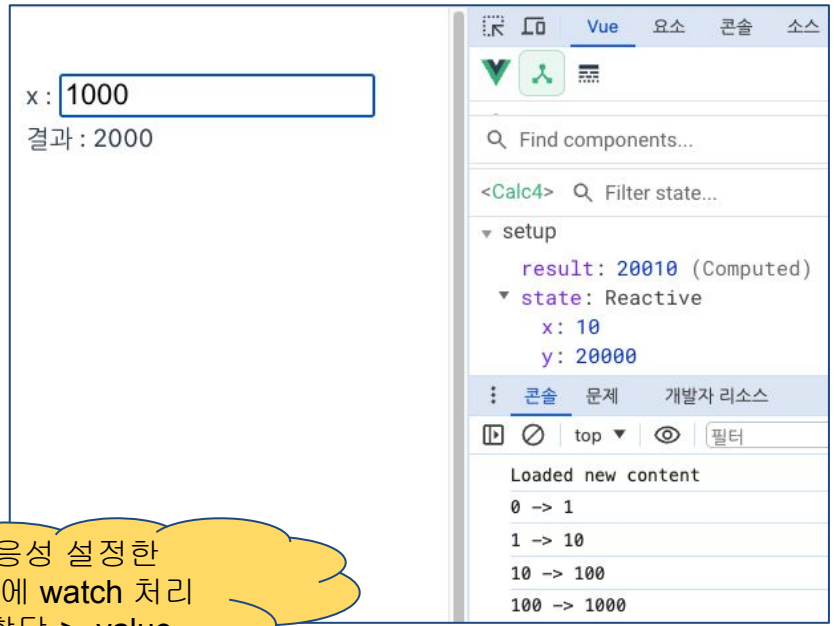
```
watch(data,      (current, old) => {
        //특정한 값의 변화에 대한 상황이 발생했을 때 처리하는 로직
}
```

- **data : watcher로 설정되는 데이터**
- **(current, old) : 상황이 발생했을 때 처리할 함수**
  - current : 변경하려고 하는 변경 값
  - old : 변경되기 전, 원래의 값
- **ref()로 설정된 값에 watch의 처리 결과값을 할당하는 경우 .value로 접근함.**

watch

Calc5.vue

```
<template>
    <div>
        x : <input type="text" v-model.number="x" /><br />
        결과 : {{result}}
    </div>
</template>

<script>
import { ref, watch } from 'vue';

export default {
    name:"Calc5",
    setup () {
        const x = ref(0);
        const result = ref(0);
        watch(x, (current, old)=>{
            console.log(`${old} → ${current}`);
            result.value = current * 2;
        })
        return { x, result }
    }
}
</script>
```

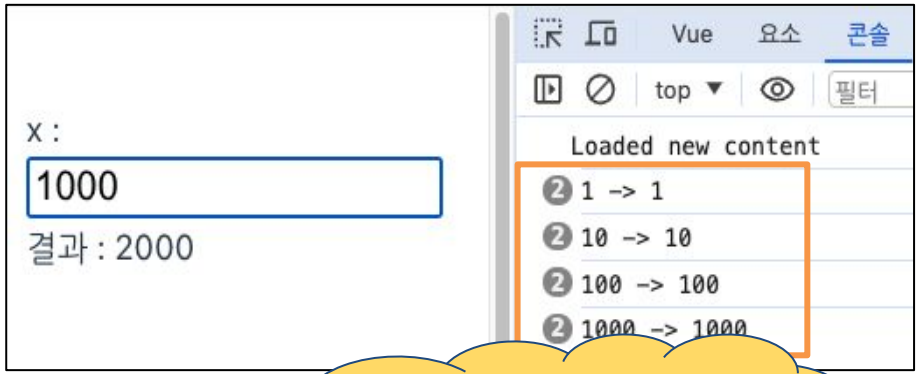


ref() 반응성 설정한
result변수에 watch 처리
결과값 할당-> .value

watch

- **reactive()로 객체 watch설정시 명확하게 설정해야함.**
- **명확하지 않으면 불필요한 실행이 발생함.**

```vue
<template>
    <div>
        x : <input type="text" v-model.number="state.x" /><br />
        결과 : {{state.result}}
    </div>
</template>

<script>
import { reactive, watch } from 'vue';

export default {
    name:"Calc6",
    setup () {
        const state = reactive({ x:0, result:0 });
        watch(state, (current, old)=>{
            console.log(`${old.x} → ${current.x}`);
            state.result = current.x * 2;
        })
        return { state }
    }
}
</script>
```

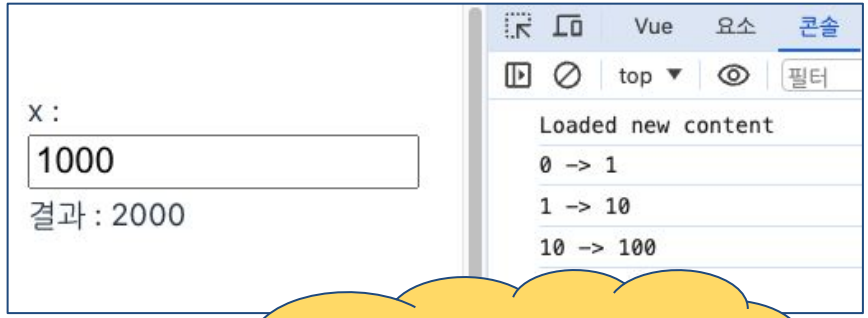


state를 감시대상자로
설정해 불필요한
호출이 일어남.

watch

- **reactive()로 객체 watch설정시 명확하게 설정해야함.**
- **명확하지 않으면 불필요한 실행이 발생함.**

```
<template>
    <div>
        x : <input type="text" v-model.number="state.x" /><br />
        결과 : {{state.result}}
    </div>
</template>

<script>
import { reactive, watch } from 'vue';

export default {
    name:"Calc6",
    setup () {
        const state = reactive({ x:0, result:0 });
        watch(()=>state.x, (current, old)=>{
            console.log(`${old} → ${current}`);
            state.result = current * 2;
        })
    }
}
</script>
```



state.x를 watch로 설정해 필요한 호출만 일어남.

state.x 수정
() => state.x

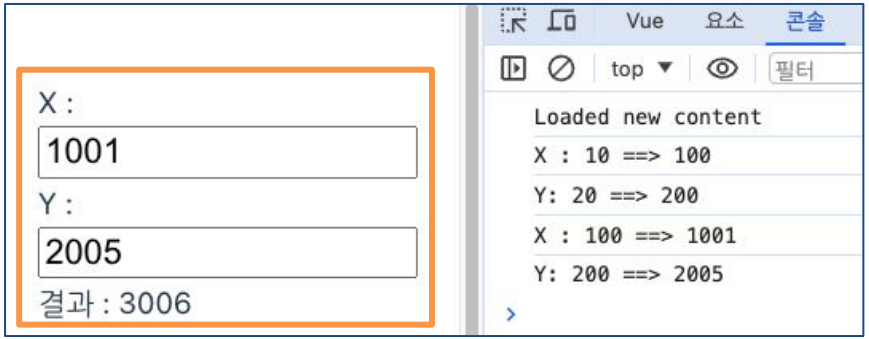
```
watch(state.x, (current, old)=>{
    console.log(`${old} → ${current}`);
    state.result = current * 2;
})
```

오류

watch

- 여러 개 값 **watch**로 설정 가능

```
<template>
    <div>
        X : <input type="text" v-model.number="x" /><br/>
        Y : <input type="text" v-model.number="y" /><br/>
        <div>결과 : {{result}}</div>
    </div>
</template>

<script>
import { ref, watch } from 'vue'

export default {
    name : "Calc7",
    setup() {
        const x = ref(10);
        const y = ref(20);
        const result = ref(30);

        watch([x, y], ([currentX, currentY], [oldX, oldY] )=>{
            if (currentX !== oldX) console.log(`X : ${oldX} ⟹ ${currentX}`)
            if (currentY !== oldY) console.log(`Y: ${oldY} ⟹ ${currentY}`)
            result.value = currentX + currentY;
        })
        return { x, y, result }
    }
}
</script>
```



watch

- **watch vs. watchEffect**

구분	watch	watchEffect
감시 대상(의존성) 지정	필요. 감시 대상 데이터가 변경되면 핸들러 함수 실행	불필요. 핸들러 함수 내부에서 이용하는 반응성 데이터가 변경되면 함수가 실행
변경전 값	이용 가능	이용 불가. 핸들러 함수 인자 없음.
감시자 설정 후 즉시 실행 여부	즉시 실행	

watchEffect

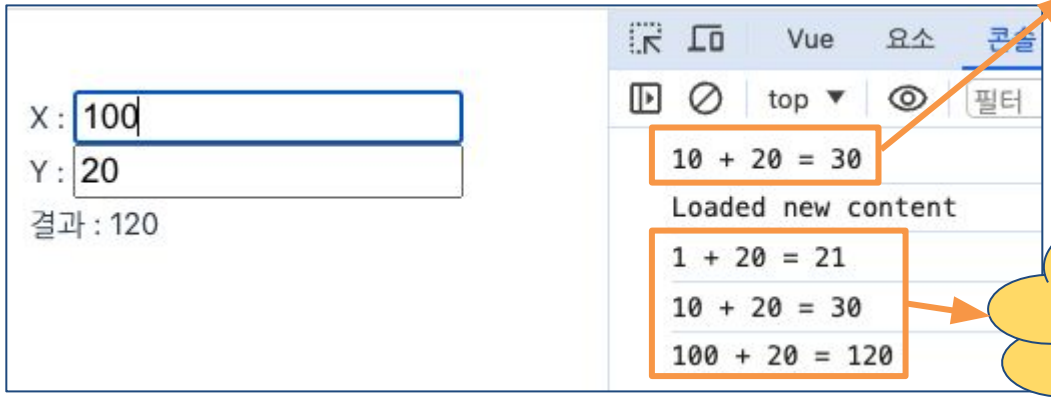
- 처음 시작시 바로 watchEffect함수가 실행
- 함수 내 변수들을 감시할 대상으로 자동 등록
- 감시대상자의 변화가 있을 때 마다 핸들러 함수 자동 실행

watch

- **watchEffect**
  - watch, watchEffect 해제도 가능
  - watchEffect 할당한 변수명() 함수 이용해 해제

handler = watchEffect();

handler() → watch해제



시작하자 마자
watchEffect() 자동 실행
watch대상자 자동 설정

watch대상자에
변화가 있을 때
핸들러 함수 자동
호출

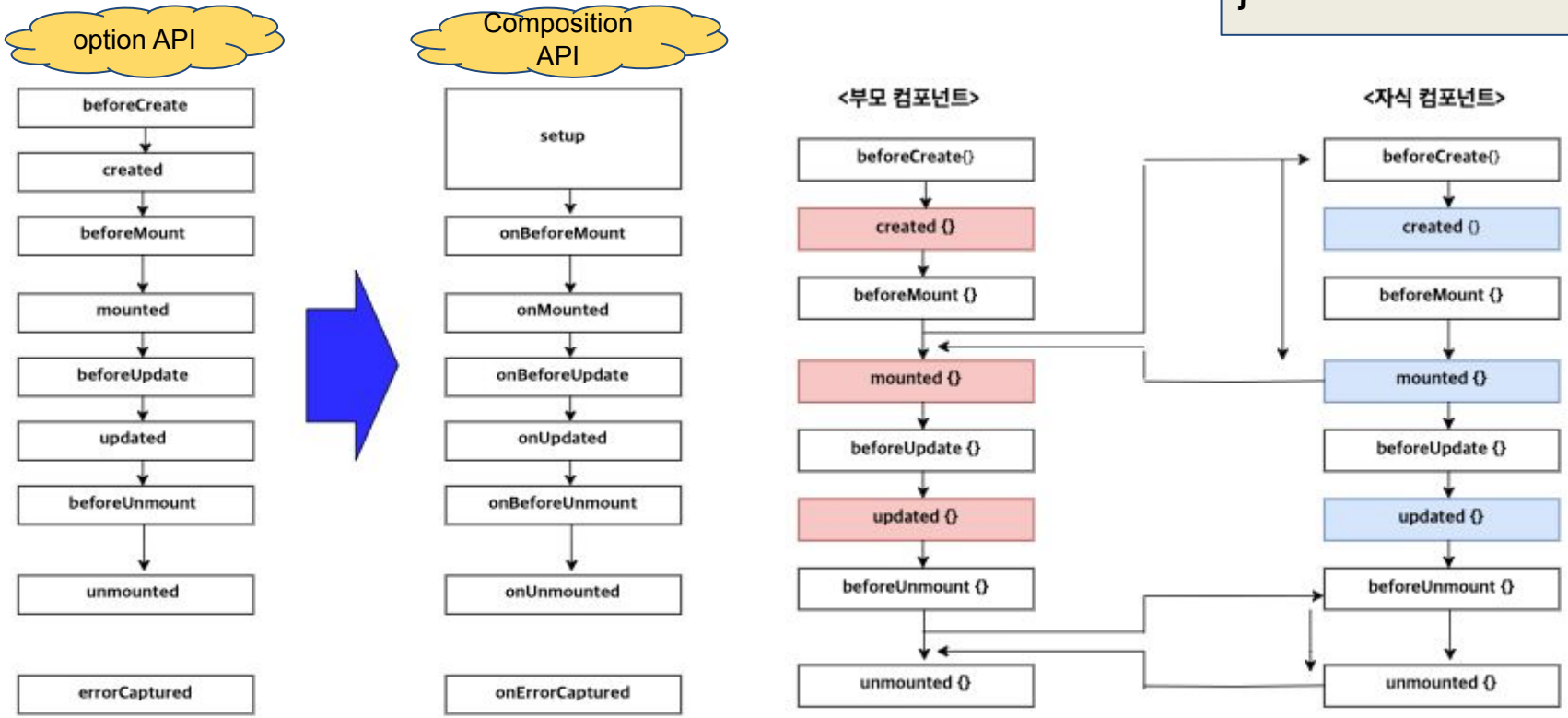
생명주기 훅(hook)

생명주기 훅

만약, Jquery를 써야한다면
어느 레벨에 넣어야할까?

- **Composition API에서도 각 생명주기에 맞는 함수를 setup()함수에**

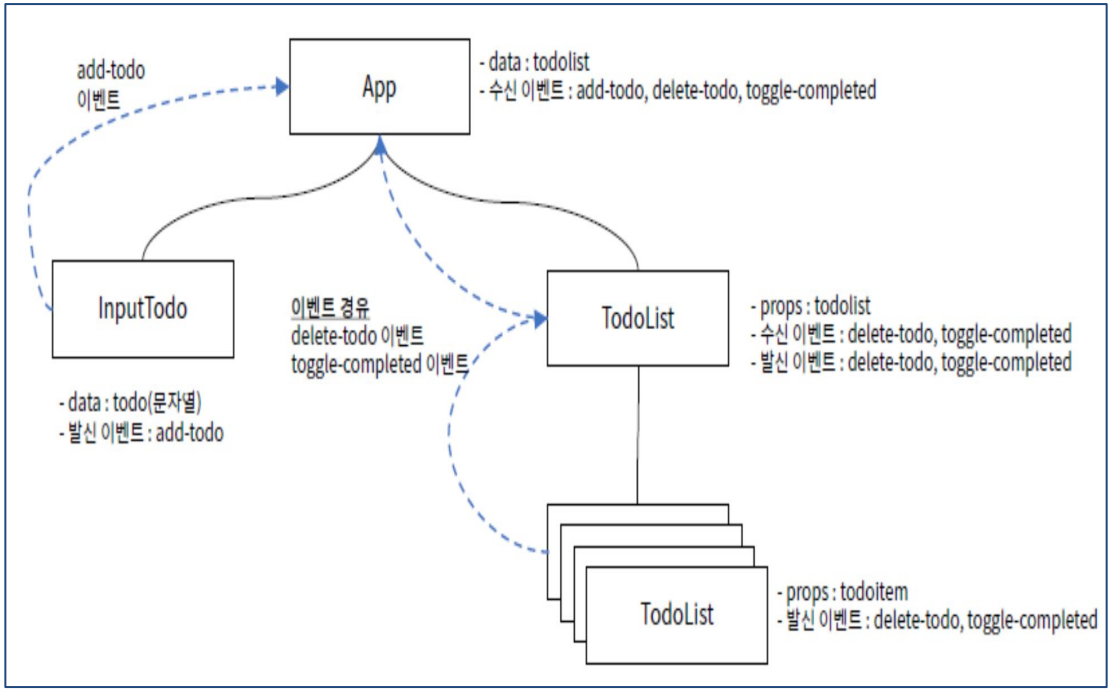
```
setup () {
  onMounted(() => {  });
}
```

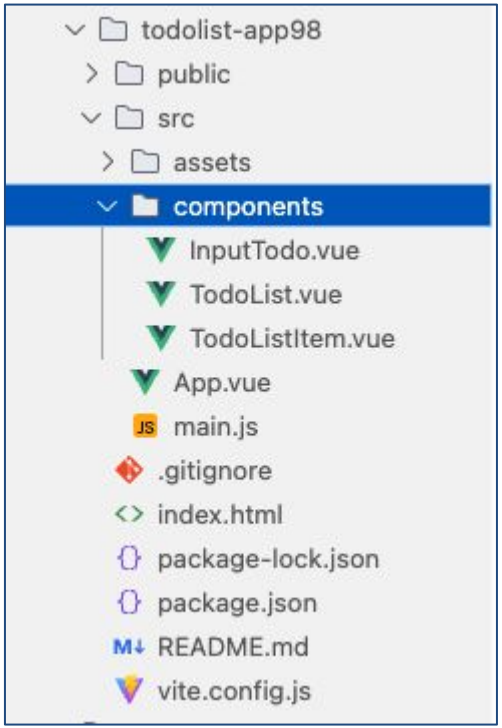


todoList 리팩토링

todoList

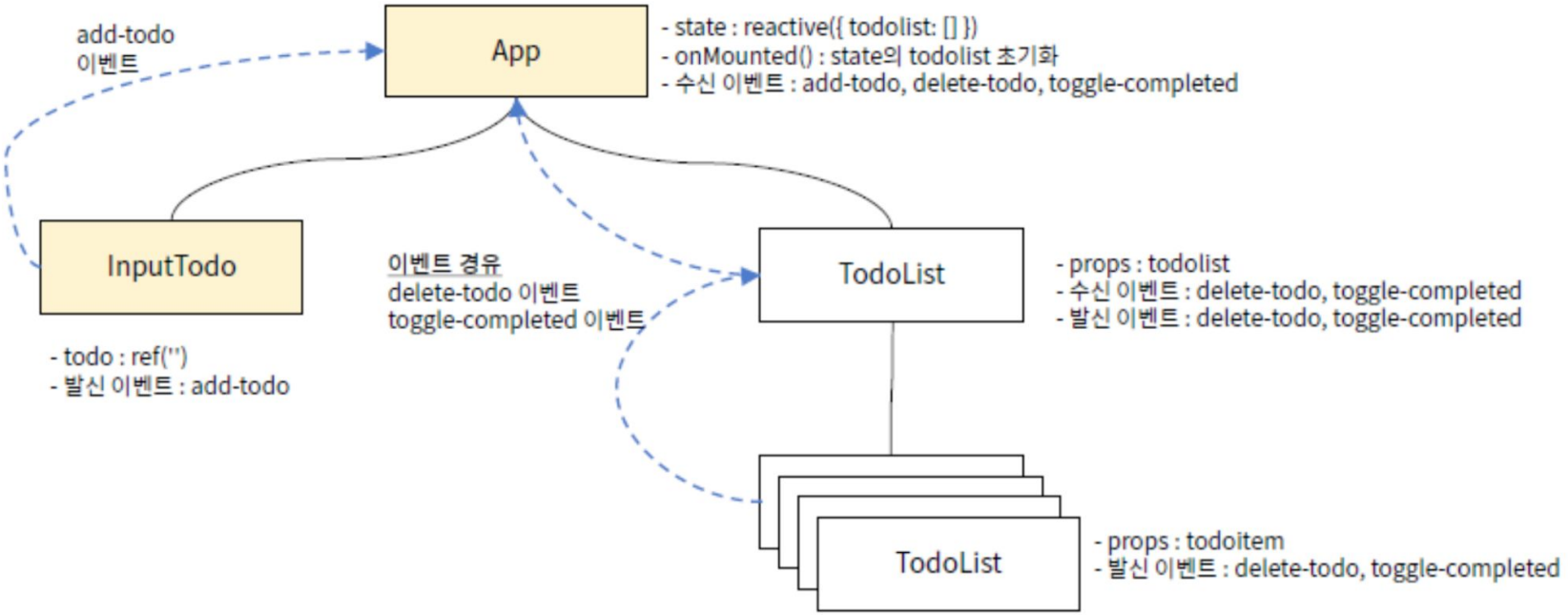
- 기존 구조





todoList

- 리펙토링 구조



todoList

App.vue

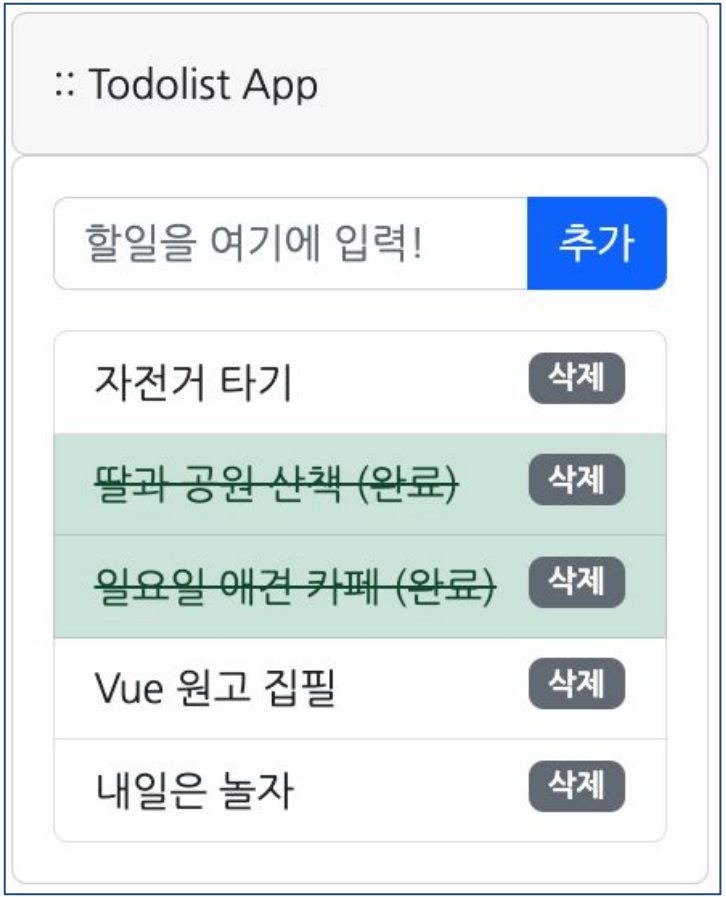
```vue
<template>
  <div class="container">
      </div>
      <div class="card card-default card-borderless">
          <div class="card-body">
              <InputTodo @add-todo="addTodo"></InputTodo>
              <TodoList :todoList="state.todoList" @delete-todo="deleteTodo"
                  @toggle-completed="toggleCompleted"></TodoList>
          </div>
      </div>
  </div>
</template>

<script>
import { reactive, onMounted } from 'vue'
import InputTodo from './components/InputTodo.vue'
import TodoList from './components/TodoList.vue'

export default {
  name : "App",
  components : { InputTodo, TodoList },
  setup() {
    const ts = new Date().getTime();
    const state = reactive({ todoList : [] })
    onMounted(()=>{
      state.todoList.push({ id: ts, todo:"자전거 타기", completed: false })
      state.todoList.push({ id: ts+1, todo:"딸과 공원 산책", completed: true })
      state.todoList.push({ id: ts+2, todo:"일요일 애견 카페", completed: false })
      state.todoList.push({ id: ts+3, todo:"Vue 원고 집필", completed: false })
    })
    const addTodo = (todo)=> {…
    }
    const deleteTodo = (id) => {…
    }
    const toggleCompleted = (id) => {…
    }
    return { state, addTodo, deleteTodo, toggleCompleted }
  }
}
</script>
```

(property) toggleCompleted: (id: any) ⇒ void

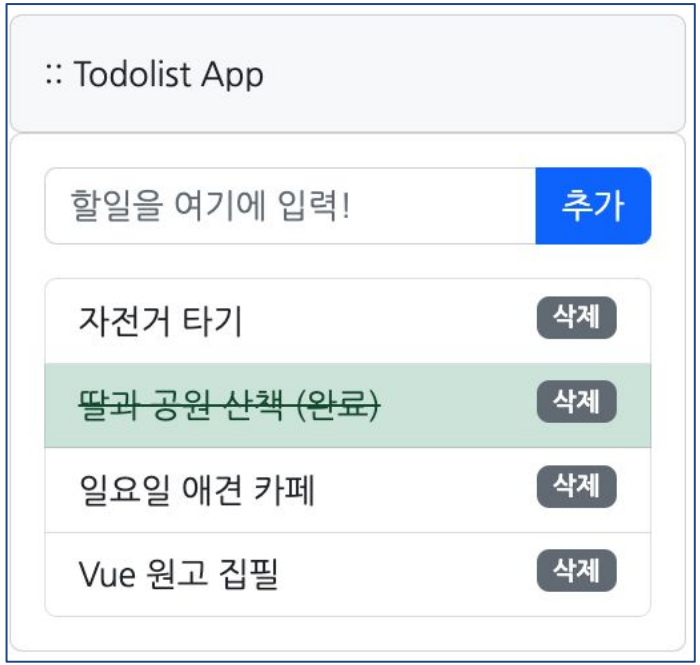
todoList

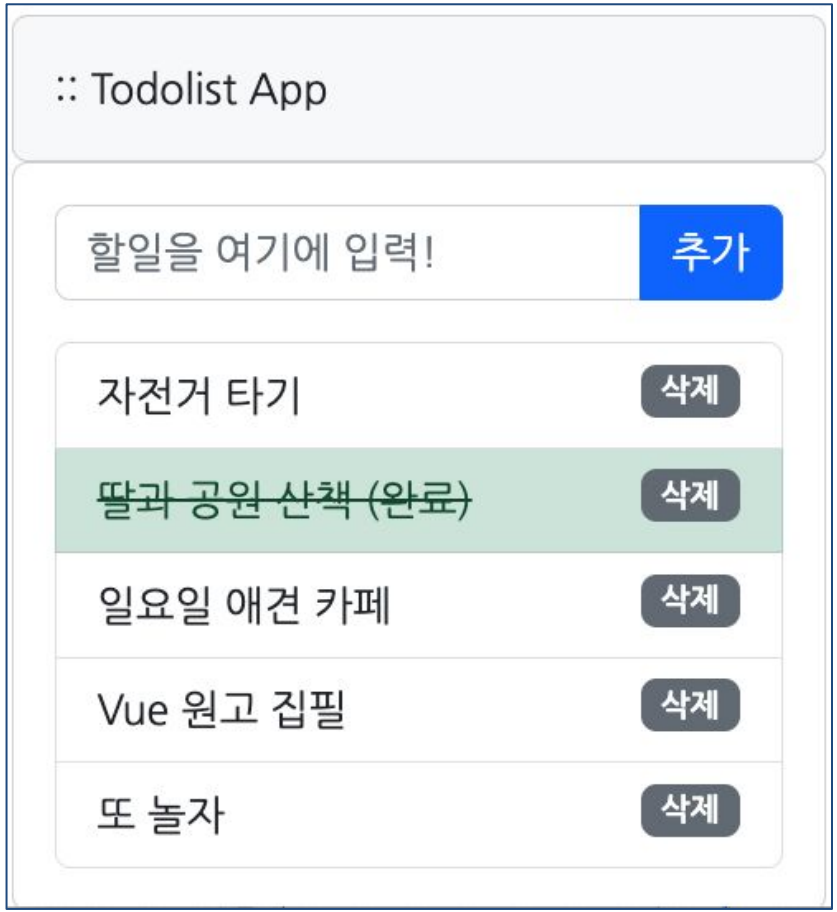
InputTodo.vue

```vue
<script>
import { ref } from 'vue'

export default {
    name : "InputTodo",
    setup(props, context) {
        const todo = ref("");
        const addTodoHandler = () => {
            if (todo.value.length >= 3) {
                context.emit('add-todo', todo.value);
                todo.value = "";
            }
        }
        return { todo, addTodoHandler }
    }
}
</script>
```



todoList


:: Todolist App
할일을 여기에 입력!
추가
자전거 타기
삭제
딸과 공원 산책 (완료)
삭제
일요일 애견 카페
삭제
Vue 원고 집필
삭제


:: Todolist App
할일을 여기에 입력!
추가
자전거 타기
삭제
딸과 공원 산책 (완료)
삭제
일요일 애견 카페
삭제
Vue 원고 집필
삭제
또 놀자
삭제

todoList

- **<script setup>으로 변경**
- **Composition API를 편하게 쓰기 위한 문법**
  - 간결하고 더 좋은 성능
  - 스크립트 내에서 최상위 선언된 변수, 작성된 변수, 함수는 리턴하지 않고도 템플릿에서 바로 사용 가능
  - components를 등록하지 않아도 괜찮음. import한 이름 그래도 사용 가능
  - props, emit 이벤트 전달은 해당하는 함수를 이용 - defineProps(), defineEmits()

< setup() >

```
setup(props, context){
        context.emit('add-todo', todo)
}
```

< script setup >

```
const props = defineProps({
        todoItem : { type:Object, required: true }
})
const emit = defineEmit(['delete-todo', 'toggle-completed'])

emit('delete-todo', id)
```

todoList

App.vue

```
<script setup>
import { reactive, onMounted } from 'vue'
import InputTodo from './components/InputTodo.vue'
import TodoList from './components/TodoList.vue'

const ts = new Date().getTime();
const state = reactive({ todoList : [] })

onMounted(()=>{
  state.todoList.push({ id: ts, todo:"자전거 타기", completed: false })
  state.todoList.push({ id: ts+1, todo:"딸과 공원 산책", completed: true })
  state.todoList.push({ id: ts+2, todo:"일요일 애견 카페", completed: false })
  state.todoList.push({ id: ts+3, todo:"Vue 원고 집필", completed: false })
})

const addTodo = (todo)=> {…
}
const deleteTodo = (id) => {…
}
const toggleCompleted = (id) => {…
}
</script>
```

components등록
삭제

return 삭제 → 스크립트 내 변수/함수 모드 템플릿에서 바로 사용 가능

todoList

InputTodo.vue

```vue
<script setup>
import { ref } from 'vue'

const emit = defineEmits(['add-todo'])
const todo = ref("");
const addTodoHandler = () => {
    if (todo.value.length >= 3) {
        emit('add-todo', todo.value);
        todo.value = "";
    }
}
</script>
```

TodoList.vue

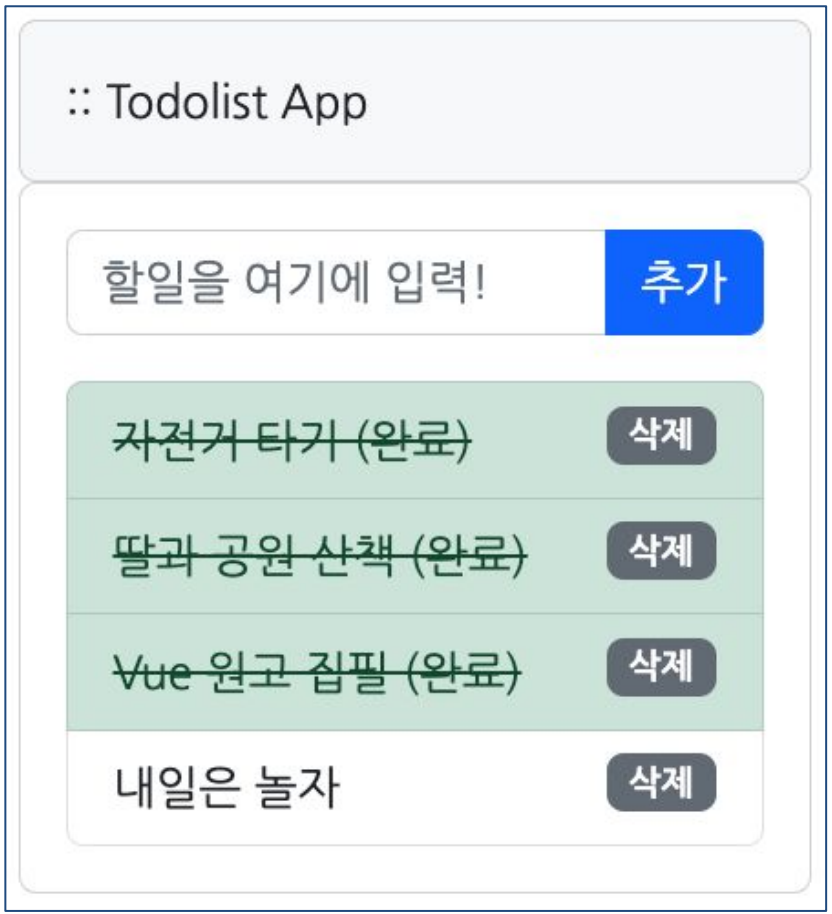
```vue
<template>
    <div class="row">
        <div class="col">
            <ul class="list-group">
                <TodoListItem v-for="todoItem in todoList" :key="todoItem.id"
                    :todoItem="todoItem" @delete-todo="emit('delete-todo', todoItem
                    @toggle-completed="emit('toggle-completed', todoItem.id)" />
            </ul>
        </div>
    </div>
</template>

<script setup>
    import TodoListItem from './TodoListItem.vue'

    const props = defineProps({
        todoList : { type : Array, required:true }
    })
    const emit = defineEmits(['delete-todo','toggle-completed'])
</script>
```

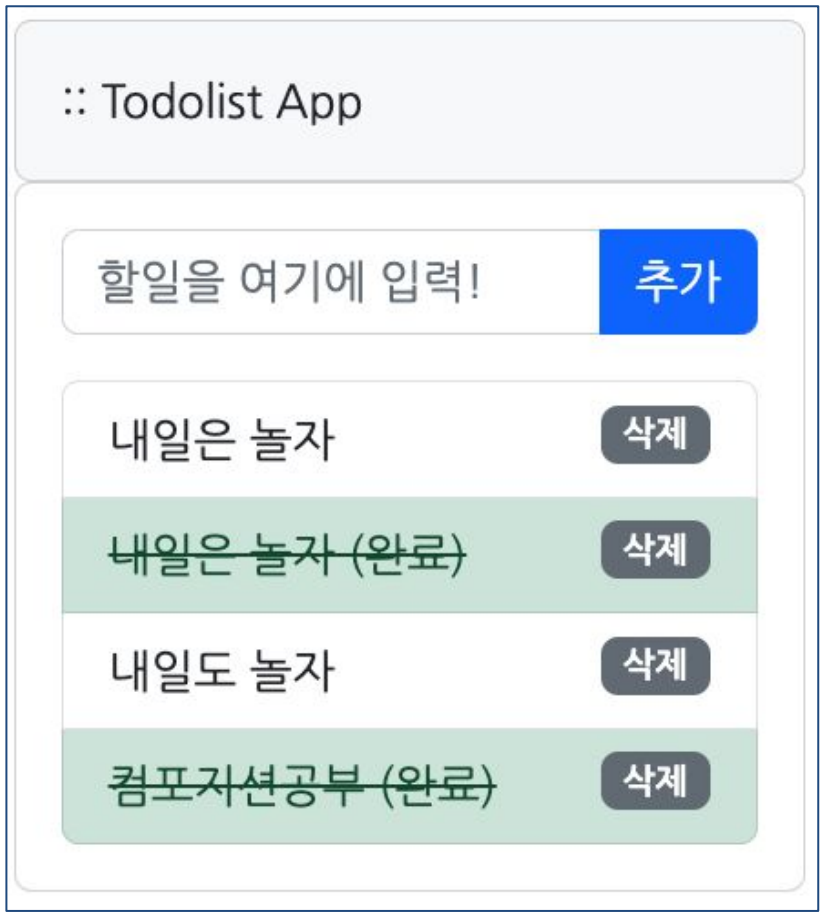
todoList

TodoListItem.vue

```
<template>
    <li class="list-group-item"
        :class="{ 'list-group-item-success': todoItem.completed } "
        @click="emit('toggle-completed', todoItem.id)" >
        <span class="pointer" :class="{ 'todo-done': todoItem.completed }">
            {{todoItem.todo}} {{ todoItem.completed ? "(완료)" : "" }}
        </span>
        <span class="float-end badge bg-secondary pointer"
            @click.stop="emit('delete-todo', todoItem.id)">삭제</span>
    </li>
</template>

<script setup>
    const props = defineProps({
        todoItem : { type : Object, required: true }
    })
    const emit = defineEmits(['delete-todo','toggle-completed'])
</script>
```

todoList


:: Todolist App
할일을 여기에 입력!
추가
자전거 타기 (완료)
삭제
딸과 공원 산책 (완료)
삭제
Vue 원고 집필 (완료)
삭제
내일은 놀자
삭제


:: Todolist App
할일을 여기에 입력!
추가
내일은 놀자
삭제
내일은 놀자 (완료)
삭제
내일도 놀자
삭제
컴포지션공부 (완료)
삭제

핵심 정리

- **Composition API**
  - Option API와 비교
- **setup**
  - ref
  - reactive
- **watch**
  - setup내에서의 watch사용법
  - watch사용하면 좋은 점 - 변경 전, 변경 후 값을 처리에 사용
  - ref로 선언된 변수를 watch에서 접근하는 방법(.value사용)
  - watch로 했을 때 감시대상 선정은 명확히 해야함.
  - 여러 개 값 감시대상로 설정 가능함.
  - watch와 watchEffect 구분, watchEffect가 좀더 용이
- **생명주기 훅**
  - setup() 내에 생명주기 훅 정의해주면 됨.