Vue JS

To Create a vue js project :

```
npm create vite@latest
app name
vue
javascript
cd 'file_name'
npm install
npm run dev

For Production Level : npm run build
```

State Management :

Vue.js-এ **state management** একটি গুরুত্বপূর্ণ অংশ, কারণ এটি একটি অ্যাপ্লিকেশনের বিভিন্ন কম্পোনেন্টের মধ্যে ডেটা পরিচালনা করা সহজ করে।

কেন state management প্রয়োজন?

1. **ডেটার শৃঙ্খলতা বজায় রাখা:**
   যখন আপনার অ্যাপ্লিকেশন বড় হয় এবং বিভিন্ন কম্পোনেন্টের মধ্যে ডেটা শেয়ার করতে হয়, তখন এটি বিশৃঙ্খল হতে পারে। State management ডেটাকে একটি নির্দিষ্ট জায়গায় সংরক্ষণ করে এবং সেই ডেটা নিয়ন্ত্রণে সাহায্য করে।
2. **কম্পোনেন্টের মধ্যে ডেটা শেয়ারিং সহজ করা:**
   একটি কম্পোনেন্ট থেকে অন্য কম্পোনেন্টে ডেটা প্রেরণ করতে সরাসরি props এবং events ব্যবহার করলে কোড জটিল হয়ে যেতে পারে। State management এই কাজটি সহজ করে।
3. **ডেটার স্থিতি বজায় রাখা:**
   ডেটার পরিবর্তন ও আপডেটের মাধ্যমে ইউজার ইন্টারফেস পরিবর্তন করার জন্য state management গুরুত্বপূর্ণ।

State এবং UI-এর সম্পর্ক কীভাবে কাজ করে?

Vue.js-এ **state** হল সেই ডেটা যা অ্যাপ্লিকেশনের UI-র সাথে যুক্ত থাকে।

- **Reactive state:** Vue ডেটাকে ট্র্যাক করে। যদি ডেটা পরিবর্তন হয়, তাহলে Vue স্বয়ংক্রিয়ভাবে UI আপডেট করে।
- **Non-reactive variable:** সাধারণ ভ্যারিয়েবল (যেমন `let number = 100;`) Vue দ্বারা ট্র্যাক করা হয় না। ফলে UI আপডেট হয় না।

Vue.js-এ ডেটা এবং UI-র মধ্যে সংযোগ (reactivity) তৈরি করার জন্য **reactive state** ব্যবহৃত হয়। সরাসরি একটি **non-reactive variable** (যেমন `let number = 100;`) ব্যবহার করলে, Vue সেই

ডেটা পরিবর্তনের খবর রাখে না এবং UI আপডেট হয় না।

```
<script setup>
let number = 100;
</script>

<template>
  <h1>{{ number }}</h1>
</template>
```

এই কোডে:

- `number` একটি সাধারণ ভ্যারিয়েবল। এটি **reactive** নয়।
- যখন আপনি UI-তে `{{ number }}` ব্যবহার করেছেন, Vue একবার মানটি UI-তে দেখাবে। কিন্তু `number` পরিবর্তিত হলে Vue এটি ট্র্যাক করবে না, তাই UI আপডেট হবে না।

সমস্যা কেন হয়?

Vue.js **reactive system** ব্যবহার করে ডেটা এবং UI-র মধ্যে সংযোগ তৈরি করে।

- সাধারণ ভ্যারিয়েবল (যেমন `let number`) Vue-র **reactivity system**-এর আওতায় আসে না।
- Vue শুধু **ref** বা **reactive** পদ্ধতির মাধ্যমে ডেটাকে ট্র্যাক করে।

Vue কেন সাধারণ ভ্যারিয়েবল ট্র্যাক করে না?

Vue **reactive system**-এ কাজ করার জন্য নির্দিষ্ট ডেটা স্ট্রাকচার ব্যবহার করে। সাধারণ ভ্যারিয়েবল:

1. **ট্র্যাকযোগ্য নয়:** Vue জানে না কখন সেই ভ্যারিয়েবলের মান পরিবর্তিত হচ্ছে।
2. **ব্যয়বহুল:** যদি Vue প্রতিটি ভ্যারিয়েবল ট্র্যাক করতে চায়, তবে অ্যাপ্লিকেশন ধীর হয়ে যাবে।

তাই Vue নির্দিষ্ট পদ্ধতির (যেমন `ref` এবং `reactive`) মাধ্যমে ডেটা ট্র্যাক করে।

Vue.js-এ state management করার জন্য দুটি জনপ্রিয় পদ্ধতি হলো:

1. Ref Method (Both support single Value and Object):

`ref` পদ্ধতিটি **reactive references** তৈরি করে। এটি সাধারণত একটি নির্দিষ্ট মানকে track করে এবং যখন সেই মান পরিবর্তন হয়, Vue স্বয়ংক্রিয়ভাবে ইউজার ইন্টারফেস আপডেট করে।

👉 কাজের প্রক্রিয়া:

- `ref` একটি reactive ডেটা তৈরি করে।

```
<script setup>

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

```
let number = ref(500);

</script>



<template>
  <h1>{{ number }}</h1>
</template>
```

```
<script setup>

import { ref } from 'vue';

let number = ref(500);

const add = () => {
  number.value = number.value + 1; // first catch the number variable and the
with a dot, catch the value of that variable -> This is the format for catching
value in ref()
}

const subtract = () => {
  number.value = number.value - 1; // first catch the number variable and the
with a dot, catch the value of that variable -> This is the format for catching
value in ref()
}

</script>

<template>
  <h1>{{ number }}</h1>
  <button @click="add"> + Plus</button>
  <button @click="subtract"> - Minus</button>
</template>
```

- যখন `number.value` পরিবর্তন হয়, Vue তা detect করে এবং DOM আপডেট করে। ম্যানুয়ালি DOM আপডেট করতে হচ্ছে না।

কিভাবে ডাটা আদান - প্রদান হচ্ছেঃ

plus/minus function কল করার সাথে সাথে number এর ভ্যালু ১ বাড়াবে/কমাবে/চেঞ্জ করবে। এই ভ্যালু সাধারণ কোনো ভ্যারিয়েবল না। এটা হল একটা state এর number। আর state এ কোনোকিছু চেঞ্জ হয়ে গেলে অটোম্যাটিক্যালি vue UI তে চেঞ্জ হয়ে যাবে। তার মানে vue UI টা রিফ্রেশ হবে/লোড নিবে একা একাই, এবং আপডেটেড ভ্যালু দেখাবে। অর্থাৎ DOM আপডেট করতে হচ্ছে না

প্রতি ক্লিকের চেঞ্জ এ। রিফ্রেশ না হলে আপডেটেড value দেখাবে না। তার মানে Memory তে অর্থাৎ console এও চেঞ্জ হয় আবার vue UI তেও। এটি হল Client Side Memory বা Computer এর Browser এর memory ।

Example with Object:

```
<script setup>

import { ref } from 'vue';

let number = ref({
  num1: 500,
  num2: 100
});



const add = () => {
  number.value.num1 = number.value.num1 + 1; // প্রথমে number.value মানে হচ্ছে স্টেট টাকে ধরা। অর্থাৎ অব্জেক্ট টাকে ধরা। এরপর . দিয়ে সেই অব্জেক্ট এর ভ্যালু টাকে ধরা। এমনটা করার কারণ হল স্টেট টি হল একটা অব্জেক্ট।
  number.value.num2 = number.value.num2 - 1;
}



const subtract = () => {
  number.value.num2 = number.value.num2 + 1; // প্রথমে number.value মানে হচ্ছে স্টেট টাকে ধরা। অর্থাৎ অব্জেক্ট টাকে ধরা। এরপর . দিয়ে সেই অব্জেক্ট এর ভ্যালু টাকে ধরা। এমনটা করার কারণ হল স্টেট টি হল একটা অব্জেক্ট।
  number.value.num1 = number.value.num1 - 1;
}

</script>

<template>

  <h1>{{ number.num1 }}</h1>
  <h1>{{ number.num2 }}</h1>

  <button @click="add"> Click1 </button>
  <button @click="subtract"> Click2 </button>

</template>
```

2. Reactive Method (Only can work with Objects):

`reactive` পদ্ধতিটি একটি **reactive object** তৈরি করে। এটি একটি জটিল ডেটা স্ট্রাকচার পরিচালনা করতে ব্যবহৃত হয়।

👉 কাজের প্রক্রিয়া:

- `reactive` একটি পূর্ণাঙ্গ অবজেক্টকে track করে।

```
<script setup>

import { reactive } from 'vue';

let number = reactive({
  num1: 500,
  num2: 100
});

const add = () => {
  number.num1 = number.num1 + 1; // যেহেতু reactive() শুধু অব্জেক্ট নিয়ে কাজ করে তাই এটিকে প্রথমে number.value ধরে এরপর . দিয়ে সেই অব্জেক্ট এর ভ্যালু টাকে ধরার প্রয়োজন পরে না। সরাসরি number.num1 অর্থাৎ object ভ্যারিয়েবলের key টাকে ধরলেই ভ্যালু টাকে ধরা যাবে।
  number.num2 = number.num2 - 1;
}

const subtract = () => {
  number.num2 = number.num2 + 1; // যেহেতু reactive() শুধু অব্জেক্ট নিয়ে কাজ করে তাই এটিকে প্রথমে number.value ধরে এরপর . দিয়ে সেই অব্জেক্ট এর ভ্যালু টাকে ধরার প্রয়োজন পরে না। সরাসরি number.num2 অর্থাৎ object ভ্যারিয়েবলের key টাকে ধরলেই ভ্যালু টাকে ধরা যাবে।
  number.num1 = number.num1 - 1;
}

</script>

<template>

  <h1>{{ number.num1 }}</h1>
  <h1>{{ number.num2 }}</h1>

  <button @click="add"> Click1 </button>
  <button @click="subtract"> Click2 </button>

</template>
```

- যখন অবজেক্টের কোন প্রপার্টি পরিবর্তন হয়, Vue তা detect করে এবং UI আপডেট করে।

Vue রিয়েক্টিভিটি এবং সরাসরি ভ্যারিয়েবল ব্যবহারের পার্থক্য

Vue-তে **reactivity system** মানে হলো:

1. **ডেটা ট্র্যাক করা**:
   Vue প্রতিটি পরিবর্তন ট্র্যাক করবে এবং জানবে কখন আপডেট করতে হবে।
2. **UI স্বয়ংক্রিয় আপডেট**:
   Reactivity system থাকলে কোনো DOM ম্যানুয়াল রেন্ডারিং করতে হবে না।

সাধারণ ভ্যারিয়েবলে এই সুবিধাগুলো নেই। আপনি সরাসরি কোডে মান আপডেট করলে Vue পুরোপুরি সেই প্রসেসে অংশ নেয় না।

```
<script setup>
let number = 100;

function changeNumber() {
  number = 110; // সাধারণ ভ্যারিয়েবল, জাভাস্ক্রিপ্ট জানে, কিন্তু Vue জানে না,
console.log() করলে console এ দেখা যাবে value চেঞ্জ গুলো, কিন্তু vue UI তে দেখা যায় না
}
</script>

<template>
  <h1>{{ number }}</h1>
  <button @click="changeNumber">Change Number</button>
</template>
```

👉 **এই ক্ষেত্রে,** `number` **পরিবর্তিত হলেও UI আপডেট হবে না, কারণ Vue এটি ট্র্যাক করছে না।** জাভাস্ক্রিপ্ট এবং ব্রাউজার কনসোল জানে যে আপনি ভ্যারিয়েবলের মান পরিবর্তন করেছেন, কারণ জাভাস্ক্রিপ্ট মেমোরি সরাসরি এই পরিবর্তন বুঝতে পারে। কিন্তু Vue.js-এর **reactivity system** সেই পরিবর্তনটি জানে না। এর ফলে Vue UI-তে সেই পরিবর্তনের প্রতিফলন হয় না।

👉 এখানে কী হবে?

- আপনি `number` এর মান পরিবর্তন করবেন, কিন্তু Vue এর **reactivity system**-এ এটি নিবন্ধিত নয়।
- তাই Vue UI আপডেট করবে না।

Reactive State ব্যবহার করে

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

const number = ref(100); // Reactive state

function changeNumber() {
        number.value = number.value + 1; // Vue জানে এটি পরিবর্তিত হয়েছে
}
</script>

<template>
```

```
        <h1>{{ number }}</h1>
        <button @click="changeNumber">Change Number</button>
</template>
```

👉 এখানে কী হবে?

- `number.value = 110;` দিলে Vue জানবে যে ডেটা পরিবর্তিত হয়েছে।
- Vue এর Proxy ট্র্যাক করবে এবং UI তৎক্ষণাৎ আপডেট করবে।

কেন এমন হয়?

1. **সাধারণ ভ্যারিয়েবলের সীমাবদ্ধতা:**
    - যখন আপনি `let number = 100;` থেকে `number = 110;` করেন, জাভাস্ক্রিপ্ট কেবলমাত্র মেমোরিতে মান আপডেট করে।
    - Vue এই ভ্যারিয়েবলকে **observe (ট্র্যাক)** করে না।
    - Vue শুধুমাত্র `ref` বা `reactive` দিয়ে তৈরি করা ডেটা ট্র্যাক করে।
2. **Vue-এর Reactivity System-এর অভাব:**
    - Vue প্রতিটি ডেটার পরিবর্তন ট্র্যাক করার জন্য Proxy ব্যবহার করে।
    - যদি আপনি সাধারণ ভ্যারিয়েবল ব্যবহার করেন, তাহলে Vue জানবে না যে এই ভ্যারিয়েবলের মান পরিবর্তন হয়েছে।
    - ফলাফল? UI-তে পরিবর্তন দেখাবে না।

কেন `ref` বা `reactive` দরকার?

Vue-তে `ref` বা `reactive` ব্যবহার করলে Vue ডেটা ট্র্যাক করে এবং স্বয়ংক্রিয়ভাবে UI আপডেট করে।

ref() and reactive(), both are Immutable (অপরিবর্তনযোগ্য):

যদি পরিবর্তন না ই হয়, তাহলে ডাটা ফ্রন্ট এন্ড এ আপডেট হয় কভাবে? এটি আসলেই পরিবর্তন যোগ্য নয়। এটি করে কি, পূর্বের ডাটা টিকে ধ্বংস করে নতুন আরেকটি ডাটা তৈরি করে এবং আগের ডাটাটিকে মুছে দিয়ে নতুন ডাটা টিকে আপডেট করে এবং ফ্রন্ট এন্ড এ ডাটাটি দৃশ্যমান হয়।

Virtual DOM কী?

Vue-তে সরাসরি **DOM (Document Object Model)**-এর সাথে কাজ করা হয় না। পরিবর্তে, Vue **Virtual DOM** ব্যবহার করে। Virtual DOM হলো একটি হালকা ওজনের জাভাস্ক্রিপ্ট অবজেক্ট যা **আসল DOM-এর ভার্চুয়াল সংস্করণ**। Vue সরাসরি আসল DOM পরিবর্তন না করে এই ভার্চুয়াল DOM-এর মাধ্যমে পরিবর্তনগুলি পরিচালনা করে।

কেন Virtual DOM ব্যবহার করা হয়?

1. **প্রদর্শনের গতি বাড়ানো:**
    - আসল DOM খুব ধীর। যেকোনো পরিবর্তন সরাসরি DOM-এ করলে অনেক সময় লাগে।

    - Virtual DOM দ্রুত পরিবর্তনগুলি গণনা করে এবং কেবলমাত্র যেটুকু প্রয়োজন সেটাই আসল DOM-এ আপডেট করে।
2. **ডেভেলপারদের জন্য সহজ:**
    - Vue-তে আমরা সরাসরি `document.getElementById()` বা `document.querySelector()` এর মতো জটিল জাভাস্ক্রিপ্ট মেথড ব্যবহার করি না।
    - বরং আমরা **ডাটার উপর ভিত্তি করে UI তৈরি করি**, আর Vue নিজেই Virtual DOM দিয়ে সব পরিচালনা করে।

Virtual DOM কীভাবে কাজ করে?

Virtual DOM আসল DOM আপডেট করার আগে তিনটি ধাপে কাজ করে:

1. **নতুন Virtual DOM তৈরি:**
   যখন কোনো ডাটা পরিবর্তন হয়, Vue নতুন Virtual DOM তৈরি করে।
2. **পুরানো এবং নতুন Virtual DOM তুলনা (Diffing Algorithm):**
   Vue পুরানো Virtual DOM এবং নতুন Virtual DOM তুলনা করে দেখে কোথায় কোথায় পরিবর্তন হয়েছে।
3. **পুনরায় রেন্ডার:**
   Vue কেবলমাত্র পরিবর্তিত অংশ আসল DOM-এ আপডেট করে।

```
<template>

  <div>
    <h1>{{ title }}</h1>
    <button @click="updateTitle">Change Title</button>
  </div>

</template>

<script setup>

import { ref } from 'vue';

const title = ref("Hello Vue!");

function updateTitle() {
  title.value = "Hello Virtual DOM!";
}

</script>
```

👉 এখানে কী হচ্ছে?

1. প্রথমে Virtual DOM তৈরি হয়, যেখানে `title` এর মান `"Hello Vue!"` ।
2. যখন আপনি বাটন ক্লিক করেন এবং `title` আপডেট করেন, Vue নতুন Virtual DOM তৈরি করে।

3. Vue দেখে কেবল `<h1>` ট্যাগে পরিবর্তন হয়েছে এবং সেই পরিবর্তিত অংশ আসল DOM-এ আপডেট করে।

Call API and Render using State:

dummy json : '[https://dummyjson.com/products'](https://dummyjson.com/products)

API কল করলে আমরা পাই একটা JSON array.

API কল করার পর ডাটা টা সরাসরি template এর ভিতরে আসে না। প্রথমে যায় State এ। এরপর যায় template এ। এই State হল একটি Media.

API কল করার পর যে ডাটা আমরা পাবো, সেটাকে আমরা State এ ঢুকাবো, এরপর সেটাকে আমরা ফ্রন্ট এন্ড এ দেখাবো।

```
<script setup>

import { ref } from 'vue';

const products = ref([]); // এই const products = ref([]); এর ref([]) হল একটি মেমোরি, যেখানে ডাটা টি স্টোর হবে।

async function getData() {
  const response = await fetch('https://dummyjson.com/products');
  const data = await response.json();

  // products.value = data; // এখানে পুরো অব্জেক্ট টাকে const products = ref([]); এর ফাকা array বা const products = ref([]); এর ref([]) মেমোরি তে ঢুকে যাবে।

  products.value = data.products; // এভাবে শুধু এর ফ্রেশ array টাকে ধরা হল
  // products.value = data['products'] // এভাবেও লেখা যায়
}

getData();

</script>

<template>

  <!-- {{ products }} এটি হল const products = ref([]); এর products ভ্যারিয়েবল -->
  <table>
    <thead>
      <tr>
        <th>ID</th>
        <th>Title</th>
        <th>Category</th>
        <th>Price</th>
      </tr>
    </thead>
```

```
    <tbody>
      <tr v-for="item in products" :key="item.id">
        <!-- এখানে products হল পুরো array এবং item হল এক একটি অব্জেক্ট।
        অর্থাৎ item দিয়ে এক একটি অব্জেক্ট কে ধরা হচ্ছে আর products দিয়ে JSON এর পুরো array টিকে বুঝাচ্ছে। -->

        <!-- <td>{{ item.id }}</td>
        <td>{{ item.title }}</td>
        <td>{{ item.category }}</td>
        <td>{{ item.price }}</td> -->

        <!-- এভাবেও লেখা যায় -->
        <td>{{ item['id'] }}</td>
        <td>{{ item['title'] }}</td>
        <td>{{ item['category'] }}</td>
        <td>{{ item['price'] }}</td>
      </tr>
    </tbody>
  </table>

</template>
```

To install Bootstrap:

npm install bootstrap

add ->
import "bootstrap/dist/css/bootstrap.css";
in main.js

same code with bootstrap

```
<script setup>

import { ref } from 'vue';

const products = ref([]); // এই const products = ref([]); এর ref([]) হল একটি মেমোরি, যেখানে ডাটা টি স্টোর হবে।

async function getData() {
  const response = await fetch('https://dummyjson.com/products');
  const data = await response.json();

  // products.value = data; // এখানে পুরো অব্জেক্ট টাকে const products = ref([]); এর ফাকা array বা const products = ref([]); এর ref([]) মেমোরি তে ঢুকে যাবে।

  products.value = data.products; // এভাবে শুধু এর ফ্রেশ array টাকে ধরা হল
```

```
  // products.value = data['products'] // এভাবেও লেখা যায়
}

getData();

</script>

<template>

  <!-- {{ products }} এটি হল const products = ref([]); এর products ভ্যারিয়েবল -->
  <div class="container">
    <div class="row">
      <div class="col-12">
        <table class="table table-striped">
          <thead>
            <tr>
              <th>ID</th>
              <th>Title</th>
              <th>Category</th>
              <th>Price</th>
            </tr>
          </thead>

          <tbody>
            <tr v-for="item in products" :key="item.id">
              <!-- এখানে products হল পুরো array এবং item হল এক একটি অব্জেক্ট।
        অর্থাৎ item দিয়ে এক একটি অব্জেক্ট কে ধরা হচ্ছে আর products দিয়ে JSON এর
পুরো array টিকে বুঝাচ্ছে। -->

              <!-- <td>{{ item.id }}</td>
        <td>{{ item.title }}</td>
        <td>{{ item.category }}</td>
        <td>{{ item.price }}</td> -->

              <!-- এভাবেও লেখা যায় -->
              <td>{{ item['id'] }}</td>
              <td>{{ item['title'] }}</td>
              <td>{{ item['category'] }}</td>
              <td>{{ item['price'] }}</td>
            </tr>
          </tbody>
        </table>
      </div>
    </div>
  </div>

</template>
```

Showing with loading progress while data fetching:

```
<script setup>

import { ref } from 'vue';

const isLoading = ref(false); // for loading animation

const products = ref([]); // এই const products = ref([]); এর ref([]) হল একটি
মেমোরি, যেখানে ডাটা টি স্টোর হবে।


getData();

async function getData() {

  //start API fetching

  isLoading.value = true; // loading animation

  const response = await fetch('https://dummyjson.com/products');
  const data = await response.json();

  // products.value = data; // এখানে পুরো অব্জেক্ট টাকে const products = ref([]);
এর ফাকা array বা const products = ref([]); এর ref([]) মেমোরি তে ঢুকে যাবে।

  // products.value = data.products; // এভাবে শুধু এর ফ্রেশ array টাকে ধরা হল
  products.value = data['products'] // এভাবেও লেখা যায়

  isLoading.value = false; // loading animation


  // Simulate a delay of 500ms for smoother user experience
  // setTimeout(() => {
  //   products.value = data['products'];
  //   isLoading.value = false; // End loading animation
  // }, 500); // loading animation

  // End API fetching
}

</script>

<template>

  <!-- {{ products }} এটি হল const products = ref([]); এর products ভ্যারিয়েবল -->


  <div class="container">
    <div class="row">
      <div class="col-12 text-center">
```

```
        <div v-if="isLoading">
          <div class="spinner-border mt-5" role="status">
            <span class="visually-hidden">Loading...</span>
          </div>
        </div>

        <div v-else>
          <table class="table table-striped">
            <thead>
              <tr>
                <th>ID</th>
                <th>Title</th>
                <th>Category</th>
                <th>Price</th>
              </tr>
            </thead>

            <tbody>
              <tr v-for="item in products" :key="item.id">
                <!-- এখানে products হল পুরো array এবং item হল এক একটি অব্জেক্ট।
        অর্থাৎ item দিয়ে এক একটি অব্জেক্ট কে ধরা হচ্ছে আর products দিয়ে JSON এর
পুরো array টিকে বুঝাচ্ছে। -->

                <!-- <td>{{ item.id }}</td>
        <td>{{ item.title }}</td>
        <td>{{ item.category }}</td>
        <td>{{ item.price }}</td> -->

                <!-- এভাবেও লেখা যায় -->
                <td>{{ item['id'] }}</td>
                <td>{{ item['title'] }}</td>
                <td>{{ item['category'] }}</td>
                <td>{{ item['price'] }}</td>
              </tr>
            </tbody>
          </table>
        </div>

      </div>
    </div>
  </div>

</template>
```

Class binding with state: