# Vue 3 composition api

# Options api nima?

Options api Vue 2 da qo’llaniladigan asosiy kod yozish stili

Dasturchilar Vue komponentga bir qancha option’larni berish orqali veb dasturlarni yaratishlari mumkin.

```
export default {
  name: 'Counter',
  data() {
    return {
      count: 0
    }
  },
  methods: {
    increment() {
      this.count++;
    },
    decrement() {
      this.count--;
    }
  }
}
```

# Options API da code reuse qilish usullari

- Mixins
- Renderless components
- Filters

Filterlar Vue 3 da mavjud emas.

# Mixin

```
const myMixin = {
  created: function () {
    this.hello();
  },
  methods: {
    hello: function () {
      console.log("hello from mixin!");
    },
  },
};
```

```
var Component = Vue.extend({
  mixins: [myMixin],
});
```

```
var component = new Component(); // => "hello from mixin!"
```

# Mixin’larning kamchiliklari

- Unclear source of properties
- Namespace’lar to’qnashuvi
- Mixin’lar aro aloqaning aniq va to’g’ridan to’g’ri emasligi

# Renderless components

```
<FetchApi
  url="https://jsonplaceholder.typicode.com/todos/1"
  v-slot="{ isLoading, data }"
>
  <div v-if="isLoading">Loading...</div>
  <div>Todo: {{ data.title }}</div>
</FetchApi>
```

Todo: delectus aut autem

# Composition API nima?

Composition API - componentda option’larni yozish o’rniga mavjud funksiyalarni import qilish va ulardan foydalanish. Bu funksiyalar composable’lar deyiladi.

Composition API quyidagilardan iborat:

- **Reactivity API:** `ref()`, `reactive()`, `computed()`, `watch()` va boshqa funksiyalar orqali reactive state hosil qilish va uni boshqarish.
- **Lifecycle hooks:** `onMounted()`, `onUnmounted()` va boshqa funksiyalar orqali Vue componentning lifecycle hooklaridan foydalanish.
- **Dependency injection:** `provide()`, `inject()`

# Composition API ga misol. setup option

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

export default {
  setup() {
    const count = ref(0);

    const increment = () => count.value++;
    const decrement = () => count.value--;

    return { count, increment, decrement };
  },
};
```

```
<template>
  <div>
    <button @click="increment">-</button>
    <span>{{ count }}</span>
    <button @click="decrement">+</button>
  </div>
</template>
```

# ref

ref asosan primitive o’zgaruvchilarni reaktiv qilish uchun ishlatiladi. Ammo obyektlar uchun ham ref ishlatish mumkin.

ref funksiyasi argument sifatida biror qiymatni qabul qiladi va mutable ref obyekt qaytaradi. Bu obyektda yagona `.value` degan property mavjud bo’ladi.

ref obyektning ichidagi `.value` property’sini o’zgartirishimiz mumkin va bu o’zgarish reaktiv bo’ladi.

```
const count = ref(1);

count.value = 10;

const users = ref(['Ali', 'Bilol']);

users.value.push('Muhammad');
```

# reactive

reactive orqali biror bir tayyor obyektni reactive qilishimiz mumkin.

```
const user = reactive({
  name: 'Ali',
  age: 30,
  gender: 'male'
});

user.name = 'Bilol';
user.age = 35;
```

# ref vs reactive

Quyidagi savollar tug’ilishi mumkin:

- Nega ref va reactive? Bitta refni o’zi yetarli emasmi?
- Nega ref ni `.value` qilib ishlatamiz, lekin reactive da to’g’ridan-to’g’ri ishlataveramiz?
- Bu ikkalasining farqi nimada va qachon qay birini ishlatish kerak?

# Composition vs Options API