A View on Vue3

What's the issue with Vue 2.0?

- Vue 2.0 came out in September 2016 -> Predating a lot of modern JS features
- Supported Typescript out of the box but with limited type interference inside vue components - Lot of manual typing required
- Multiple alternatives like Vue Class Components or Vue with TSX tried to improve Typescript handling -> Fragmentation
- Has some limitations of the reactivity system (Vue.set etc.) -> Occasional Annoyance
- Duplicate functionality (two way data-binding) -> Larger api-surface

Example Vue2 app

```
export default Vue.extend({
  components: {
    Player: () => import('./player.js'),
  },
  props: {
    player2StartValue: {
      type: Number as PropType<number>,
      default: 10,
    },
  },
  data() {
    return {
      flags: {
        // hasStarted: false,
      },
      players: [
        {
          name: 'PLAYER1',
          score: 0,
        },
        {
          name: 'PLAYER2',
          score: 0,
        },
      ],
    } as Data;
  },
  computed: {
    getTotalNumberOfClicks(): number {
      const players = this.players as Player[];

      const sumOfClicks = players.reduce((total, current) => {
        const newTotal = total + current.score;

        return newTotal;
      }, 0);

      return sumOfClicks;
    },
```

```
export default Vue.extend({
  props: {
    player: {
      type: Object as PropType<Player>,
      default: () => ({
        name: 'Ted',
        score: 0,
      }),
    },
  },
  methods: {
    handleClick(): void {
      this.$emit('update-player');
    },
  },
  template: `
    <div>
      <dl>
        <dt>{{ player.name }}</dt>
        <dd>{{ player.score }}</dd>
      </dl>
      <button
        type="button"
        @click="handleClick()"
      >
        Click
```

New features & Improvements

Composition API

- Improves compatibility with Typescript
- Improves readability/code organization
- Provides data encapsulation
- Simplifies scope handling
- Further fragmentation of the Vue ecosystem
- Changes approach to unit testing
- Still actively changing (script setup)

```
export default defineComponent({
  setup() {
    const counter = ref<number>(0);
    const hasBeenClicked = computed<boolean>(() => counter.value !== 0);

    function handleClick(): void {
      counter.value += 1;
    }

    return {
      counter,
      hasBeenClicked,
      handleClick,
    };
  },
  template: `
    <h1>
      <template v-if="hasBeenClicked">Has started</template>
      <template v-else>Has not started</template>
    </h1>
    <button
      type="button"
      @click="handleClick"
    >
      Clicked {{ counter }} time(s)
    </button>
  `,
});
```

Example Composition API

Better reactivity

- Vue3 uses proxy objects for reactivity
- Objects are now fully reactive by default without any change detection issues
- Objects can be made partially reactive
- Requires evergreen browsers (newer than IE11)
- Makes destructuring slightly more complicated

Multiple v-model attributes for two-way data-binding

- Two-way data-binding mostly used for form-elements within components and child to parent communication on custom components
- v-model and sync-modifier are used for two way data-binding in Vue2
- sync-modifier has been removed in Vue3
- Vue3 allows multiple v-models per component

```
export default defineComponent({
  props: {
    firstName: {
      type: String as PropType<string>,
      default: '',
    },
    lastName: {
      type: String as PropType<string>,
      default: '',
    },
  },
  computed: {
    firstNameValue: {
      set(newValue: string): void {
        this.$emit('update:firstName', newValue);
      },
      get(): string {
        return this.firstName;
      },
    },
  },
  methods: {
    handleInput(event: InputEvent): void {
      if (event.target === null) {
        throw new Error('input field missing');
      }
      const targetElement = event.target as HTMLInputElement;

      this.$emit('update:lastName', targetElement.value);
    },
  },
  template: `
    <div>
      <label for="first-name">First name: </label>
      <input v-model="firstNameValue" id="first-name" />
    </div>
    <div>
      <label for="last-name">Last name: </label>
      <input :value="lastName" @input="handleInput" id="last-name" />
    </div>
  `,
});
```

Event-validation and Event-documentation

- Events can now be validated via emit property
- See at a glance which events are emitted by a component and check the validity of emitted event parameters
- Can be simple string array for documentation only
- Can be an object of callback functions or validator functionality
- Eslint shows warning when events that are used are missing in the emits property

```
export default defineComponent({
  props: {
    text: {
      type: String as PropType<string>,
      default: '',
    },
  },
  emits: {
    'update-text': (newValue: string) => {
      const nonLatinCharactersRegex = /[^\x00-\x7F]+/;

      if (nonLatinCharactersRegex.test(newValue)) {
        return false;
      }

      return true;
    },
    'toggle-button': null, // disable validation
  },
  computed: {
    textValue: {
      set(newValue: string): void {
        this.$emit('update-text', newValue);
      },
      get(): string {
        return this.text;
      },
    },
  },
  methods: {
    handleButtonClick(): void {
      this.$emit('toggle-button');
    },
  },
  template: `
```

Multiple root nodes in templates

- Templates can now have multiple root elements
- No more wrapper elements necessary
- Class-attribute placement needs to be handled manually
- Key placement slightly more complex when using template-tags with v-for

```
export default defineComponent({
  data() {
    return {
      hasLoaded: false,
    };
  },
  mounted() {
    setTimeout(() => {
      this.hasLoaded = true;
    }, 5000);
  },
  template: `
    <div v-if="!hasLoaded">
      <marquee>
        Table is loading. Please wait!
      </marquee>
    </div>
    <table v-else>
      <tr>
        <th>Head 1</th>
        <th>Head 2</th>
      </tr>
      <tr>
        <td>Column 1</td>
        <td>Column 2</td>
```

Breaking changes

Filter removal

- Filter syntax was original taken from AngularJS
- Often used for output formatting of currencies, dates and strings
- Pure functions with no access to component state
- Can be replaced with regular functions or computed properties

```
export default Vue.extend({
  filters: {
    capitalizeLastLetters(value: string | null): string {
      if (!value) {
        return '';
      }

      const words: string[] = value.trim().split(/\s+/);
      const wordsReversed: string[] = words
        .map((word) => word.split('').reverse().join(''));

      const wordsReversedCapitalized: string[] = wordsReversed.map((word) => {
        const firstLetter = word.charAt(0).toUpperCase();
        const remainingLetters = word.slice(1).toLowerCase();
        const newWord = `${firstLetter}${remainingLetters}`;

        return newWord;
      });

      const wordsCapitalized: string[] = wordsReversedCapitalized
        .map((word) => word.split('').reverse().join(''));
      const wordsCapitalizedString: string = wordsCapitalized.join(' ');

      return wordsCapitalizedString;
    },
  },
  data() {
    return {
      text: '',
    };
  },
  template: `
    <div class="root">
      <input v-model="text" placeholder="Please enter a text" />
      <p>
        Filtered: {{ text | capitalizeLastLetters }}
      </p>
    </div>
  `,
});
```

Example filters Vue2

v-if & v-for precedence swap

- Using both directives at the same time should be avoided and will result in a linter warning
- Vue2 executes **v-for** first and creates a temporary object which is then used by **v-if** to check if the condition is fulfilled.
- Vue3 executes **v-if** first and doesn't render anything as there is no age variable in scope, which causes a warning
- Can be refactored into a computed property that filters and caches the results
- For conditional rendering an additional computed property like hasResults can be used

```
export default Vue.extend({
  data() {
    return {
      ageLimit: 5,
      people: [
        {
          name: 'Loona Loonie',
          age: 1,
        },
        {
          name: 'Ted Tenner',
          age: 10,
        },
        {
          name: 'Franklin Fiver',
          age: 5,
        },
      ],
    };
  },
  computed: {
    filteredPeople() {
      const people = this.people as Person[];
      const ageLimit = this.ageLimit as number;

      return people.filter((person) => person.age >= ageLimit);
    },
  },
  template: `
    <div class="root">
      <dl v-for="person in people" v-if="person.age >= 5">
        <dt>Name: </dt>
        <dd>{{ person.name }}</dd>
        <dt>Age: </dt>
        <dd>{{ person.age }}</dd>
      </dl>
      <hr />
      <dl v-for="person in filteredPeople">
        <dt>Name: </dt>
        <dd>{{ person.name }}</dd>
```

.sync removal and v-model name changes in custom components

- .sync-attribute has been dropped in Vue3 and needs to be replaced with v-model
- Name of v-model props have changed and differ between components with single v-model and multiple v-models
- For custom components that use a single v-model attribute, the naming convention is the following:
- Name of the v-model attribute on the parent component: `v-model` or `v-model:modelValue`
- Name of the emitted event: `update:modelValue`
- V-model prop name: `modelValue`

```ts
const radioButtonStates = {
  on: 'On',
  off: 'Off',
} as const;
type RadioButtonStates = keyof typeof radioButtonStates

export default defineComponent({
  props: {
    modelValue: {
      type: String as PropType<RadioButtonStates>,
      default: 'off',
    },
  },
  emits: [
    'update:modelValue',
  ],
  computed: {
    modelValueComputed: {
      set(newValue: RadioButtonStates): void {
        this.$emit('update:modelValue', newValue);
      },
      get(): string {
        return this.modelValue;
      },
    },
  },
```

Example single v-model

.sync removal and v-model name changes in custom components

- For custom components that use multiple v-model attributes, the naming convention is the following:
- Name of the v-model attribute on the parent component: `v-model:nameOfVariable`
- Name of the emitted event: `update:nameOfVariable`
- V-model prop name: `nameOfVariable` not `modelValue:nameOfVariable`

```
export default defineComponent({
  props: {
    firstName: {
      type: String as PropType<string>,
      default: '',
    },
    lastName: {
      type: String as PropType<string>,
      default: '',
    },
  },
  computed: {
    firstNameValue: {
      set(newValue: string): void {
        this.$emit('update:firstName', newValue);
      },
      get(): string {
        return this.firstName;
      },
    },
  },
  methods: {
    handleInput(event: InputEvent): void {
      if (event.target === null) {
        throw new Error('input field missing');
      }
      const targetElement = event.target as HTMLInputElement;

      this.$emit('update:lastName', targetElement.value);
    },
  },
  template: `
    <div>
      <label for="first-name">First name: </label>
      <input v-model="firstNameValue" id="first-name" />
    </div>
    <div>
      <label for="last-name">Last name: </label>
      <input :value="lastName" @input="handleInput" id="last-name" />
    </div>
  `,
});
```

Other minor breaking changes

- Recommended naming conventions for event-names has changed
- Key placement on multi-root components
- Template tag rendering
- Browser support

The end