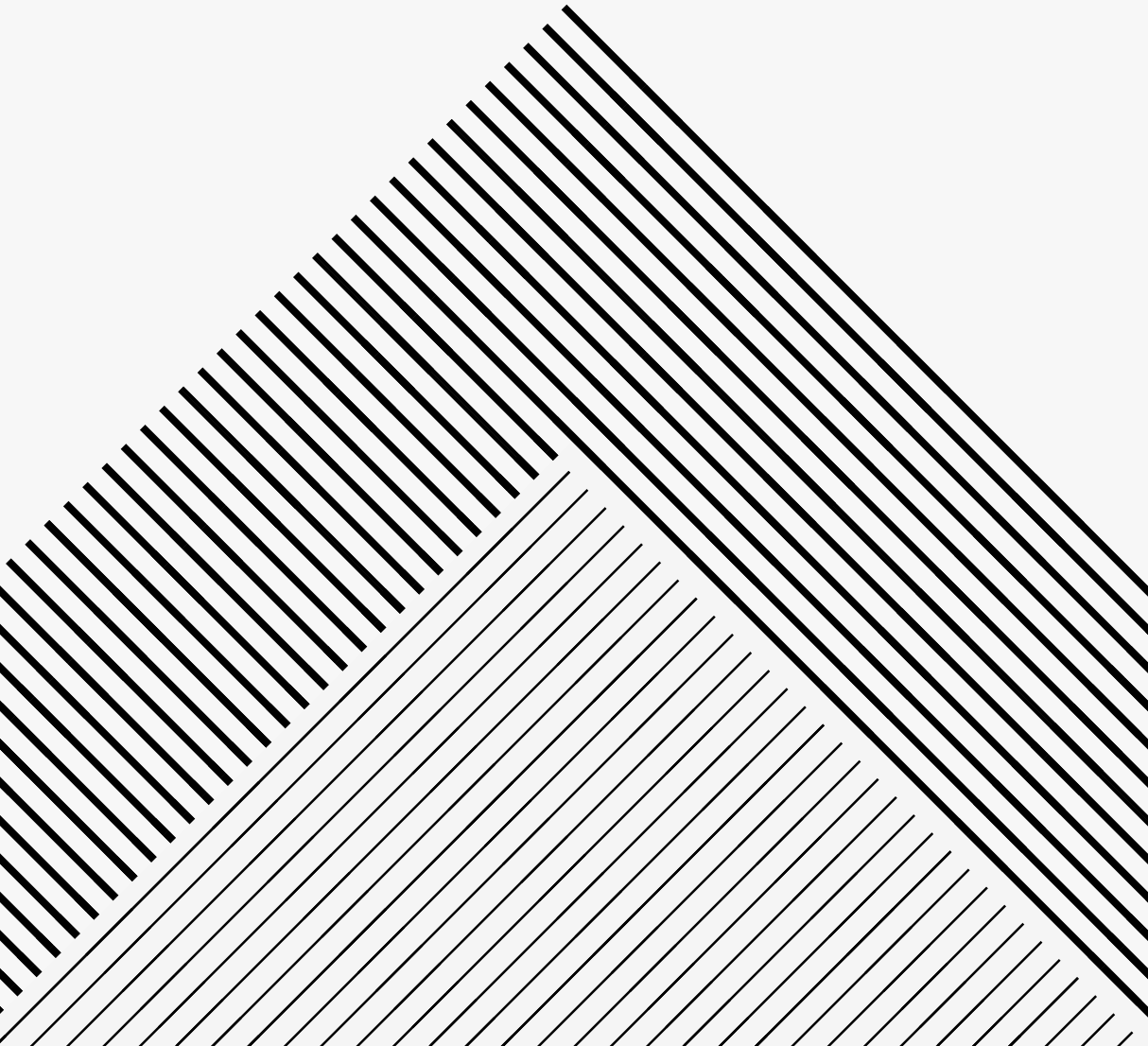


Note

1. **In index.html**

<script src="https://unpkg.com/vue@next"></script>

2. **Example of structure**

```
Vue.createApp({
  data() {
    return {
      goals: [],
      enteredValue: "",
    };
  },
  methods: {
    addGoal() {
      this.goals.push(this.enteredValue);
      this.enteredValue = "";
    },
  },
}).mount("#app");
```

mount: indicate the part controlled by Vue.

3. **Output data**

{{ courseGoal }}

Learn more <a v-bind:href="vueLink">about Vue.

4. **Call Function**

{{ outputGoal() }}

5. **Output Html element**

courseGoalB: '<h2>Master Vue and build amazing apps !</h2>',

6. **EventListener**

<button v-on:click="add( )">Add</button>

7. **Native Event Object**

<input type="text" v-on:input="setName">

```
setName(event) {
    this.name = event.target.value;
}
```

<input type="text" v-on:input="setName($event, 'Wang')">

```
setName(event, lastName) {
    this.name = event.target.value + ' ' + lastName;
}
```

8. **Event modifier**

- <form v-on:submit.prevent="submitForm">

  equals

  event.preventDefault(); // prevent webpage from reloading

- <input type="text"

  v-on:input="setName($event, 'Wang')"

  v-on:keyup.enter="confirmInput"> // to confirme the input

- <p v-once>Starting Counter: {{ counter }}</p> //the content will not be changed

9. **Two-way Binding for INPUT**

v-model = v-bind:value + v-on:input

10. **Computed properties**

```
const app = Vue.createApp({
  data() {…
  },
  computed: {
    fullname() {
      console.log('running again...');
      if(this.name === ''){
        return '';
      }
      return this.name + ' ' + 'WANG';
    }
  },
  methods: {…
  }
});
```

• used as data property (no parentheses)

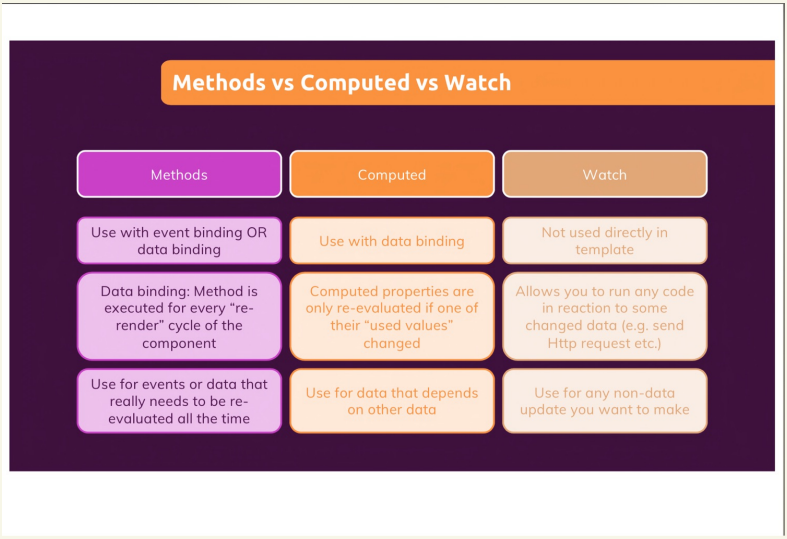
• only re-execute and re-evaluate when the dependency (here : this.name) changes

• used to calculate some output value dynamically

11. Watcher

```
data() {
  return {
    counter: 10,
    name: '',
    confirmedName: ''
  };
},
watch: {
  counter(value) {
    if(value > 50) {
      this.counter = 0;
    }
  }
},
```

• update some data property in reaction to a property changing



12. Shorthands

v-on:click —> @click

v-bind:value —> :value

13. Dynamic Styling

- <div class="demo" :style="{borderColor: boxASelected ? 'red' : '#ccc'}" @click="boxSelected('A')">
- <p :style="{backgroundColor: colorInput}">Style me inline!</p>

14. Add class Dynamically

- <div class="demo" :class="{active: boxASelected}" @click="boxSelected('A')">
- <div class="demo" :class="boxBClasses" @click="boxSelected('B')">

```
computed: {
    boxBClasses() {
        return {active: this.boxBSelected};
    }
},
```

- <div :class="['demo', {active: boxCSelected}]" @click="boxSelected('C')">

15. v-if, v-else-if, v-else

- control Html elements attaching or detaching 出现或者消失
- Elements with v-else-if or v-else must be directly after element with v-if

16. v-show instead of v-if

v-show: show or hide elements

v-if: add or remove elements of DOM

17. render list of data v-for

<li v-for="goal in goals">{{ goal }}

<li v-for="(goal, index) in goals">{{ goal }} - {{ index }}

<li v-for="(key, value, index) in {name: 'biyun', age: '25'}">{{ key }} - {{ value }} - {{ index }}

18. Array. splice( )

array.splice(starting index, number of elements to remove, elements to add ...)

// supprime 2 éléments à partir de l'index 0, et insère "perroquet", "anémone" et"bleu"

ex: mesPoissons.splice(0, 2, "perroquet", "anémone", "bleu");

19. List & key

```
<li v-for="(goal, index) in goals" :key="goal" @click="removeGoal(index)">
    <p>{{ goal }} - {{ index }}</p>
    <input type="text" @click.stop/>     //stop the event to remove goal
</li>
```

In Vue, when we remove the first list element, Vue won't remove it actually, but replace the dynamic content of seconde element to the first one. If the list element include a child element like input, input element is still the old first element, so we will lose the input we had on the seconde element. So we need a key attribute to identify every list element.

Monster Project

20. Math

- calculate a random value between 5 and 12:

```
Math.floor( Math.random( ) * (12 - 5)) + 5;
```

求一个小于此浮点数的最大整数

21. Array

```
Array.push( )      //add the element on the end
Array.unshift( )   // add the element on the beginning
```

22. Refs

```
<input type="text" @input="saveInput">
```

```
saveInput(event) {
  this.currentUserInput = event.target.value;
},
```

Input event log what the user enters with every keystroke

```
<input type="text" ref="userText">
```

```
setText() {
  // this.message = this.currentUserInput;
  this.message = this.$refs.userText.value;
},
```

Ref attribute permit accessing the input value when we ask

23. detect click outside element: tabindex = 0

```
<ul v-if="optionsIsVisible"
@focusout="Close('option')"
tabindex="0">
```

23. Components

when a HTML page has blocks duplicated, and each has a event independent , in this case, v-for cannot satisfy. We should use component.

Développement Server

1. npm install -g @vue/cli
2. vue create vue-first-app
3. cd vue-first-app
4. npm run serve

1. Adding Components

1.1. Component independent

```
<template>
    <li>
        <h2>{{ friend.name }}</h2>
        <button @click="toggleDetails">{{ detailsAreVisible ? 'Hide' : 'Show'}}
        details</button>
        <ul v-if="detailsAreVisible ">
            <li><strong>Phone:</strong> {{ friend.phone }}</li>
            <li><strong>Email:</strong> {{friend.email }}</li>
        </ul>
    </li>
</template>

<script>
export default {
    data() {
        return {
            detailsAreVisible: false,
            friend: {
                id: "manuel",
                name: "Manuel Lorenz",
                phone: "01234 5678 991",
                email: "manuel@localhost.com",
            }
        }
    },
    methods: {
        toggleDetails() {
            this.detailsAreVisible = !this.detailsAreVisible;
        }
    },
};
</script>
```

1.2. main.js

```
import FriendContact from './components/FriendContact.vue';

const app = createApp(App);
app.component('friend-contact', FriendContact);
app.mount('#app');
```

1.3. App.vue

```
<ul>
    <friend-contact></friend-contact>
    <friend-contact></friend-contact>
</ul>
```

2. Components Communication (parent => child)

2.1. App.vue

```
<ul>
  <friend-contact
    name="Manuel Lorenz"
    phone-number="01234 5678 991"
    email-address="manuel@localhost.com"
  ></friend-contact>
  <friend-contact
    name="Julie Jones"
    phone-number="09876 543 221"
    email-address="julie@localhost.com"
  ></friend-contact>
</ul>
```

phone-number: kebab case in HTML code

2.2. FriendContact.vue

```
<template>
    <li>
        <h2>{{ name }}</h2>
        <button @click="toggleDetails">{{ detailsAreVisible ? 'Hide' : 'Show'}}
        details</button>
        <ul v-if="detailsAreVisible ">
            <li><strong>Phone:</strong> {{ phoneNumber }}</li>
            <li><strong>Email:</strong> {{ emailAddress }}</li>
        </ul>
    </li>
</template>

<script>
export default {
    props: [
        'name',
        'phoneNumber',
        'emailAddress'
    ],
```

phoneNumber: camelCase in JavaScript

3. Validating props

```
//props: ["name", "phoneNumber", "emailAddress", "isFavorite"],
props: {
  name: {
    type: String,
    required: true,
  },
  phoneNumber: {
    type: String,
    required: true,
  },
  emailAddress: {
    type: String,
    required: true,
  },
  isFavorite: {
    type: String,
    required: false,
    default: "0",
    validator: function(value) {
      return value === "1" || value === "0";
    },
  },
},
```

4. Dynamic props

4.1. App.vue

```
<ul>
  <friend-contact
    v-for="friend in friends"
    :key="friend.id"
    :name="friend.name"
    :phone-number="friend.phone"
    :email-address="friend.email"
    :is-favorite='true'
  ></friend-contact>
```

5. Emitting Custom Events (Child => parent)

5.1. FriendContact.vue

```
<template>
  <li>
    <h2>{{ name }} {{ isFavorite ? "(Favorite)" : "" }}</h2>
    <button @click="toggleDetails">
      {{ detailsAreVisible ? "Hide" : "Show" }} details
    </button>
    <button @click="toggleFavorite">
      Toggle Favorite
    </button>
    <ul v-if="detailsAreVisible">
      <li><strong>Phone:</strong> {{ phoneNumber }}</li>
      <li><strong>Email:</strong> {{ emailAddress }}</li>
    </ul>
  </li>
</template>
```

```
methods: {
  toggleDetails() {
    this.detailsAreVisible = !this.detailsAreVisible;
  },
  toggleFavorite() {
      this.$emit('toggle-favorite', this.id);
  }
},
```

5.2. App.vue

```
<template>
  <section>
    <header><h1>My Friends</h1></header>
    <ul>
      <friend-contact
        v-for="friend in friends"
        :key="friend.id"
        :id="friend.id"
        :name="friend.name"
        :phone-number="friend.phone"
        :email-address="friend.email"
        :is-favorite="friend.isFavorite"
        @toggle-favorite="toggleFavoriteStatus"
      ></friend-contact>
    </ul>
  </section>
</template>
```

```
methods: {
  toggleFavoriteStatus(friendId) {
    const identifiedFriend = this.friends.find(
      (friend) => friend.id === friendId
    );
    identifiedFriend.isFavorite = !identifiedFriend.isFavorite;
  },
},
```

6. Form

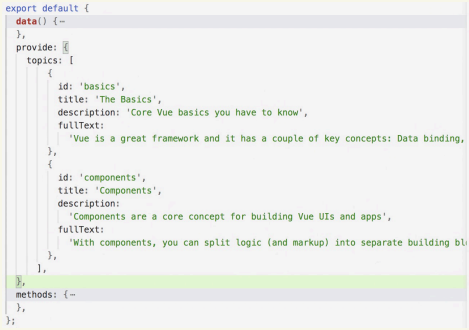
```
<template>
  <form @submit.prevent="submitData">
    <div>
      <label>Name</label>
      <input type="text" v-model="enteredName" />
    </div>
    <div>
      <label>Phone</label>
      <input type="tel" v-model="enteredPhone" />
    </div>
    <div>
      <label>E-mail</label>
      <input type="email" v-model="enteredEmail" />
    </div>
    <div>
      <button>Add Contact</button>
    </div>
  </form>
</template>
```

```vue
<script>
export default {
  emits: ["add-contact"],
  data() {
    return {
      enteredName: "",
      enteredPhone: "",
      enteredEmail: "",
    };
  },
  methods: {
    submitData() {
      this.$emit(
        "add-contact",
        this.enteredName,
        this.enteredPhone,
        this.enteredEmail
      );
    },
  },
};
</script>
```

7. Provide + Inject (data or function)

passing data within parent and child elements

7.1. App.vue



or

```vue
export default {
  data() {…
  },
  provide() {
    return {
      topics: this.topics
    }
  },
  methods: {…
  },
};
</script>
```

7.2. KnowledgeGrid.vue

```vue
export default {
  inject: ['topics'],
  emits: ['select-topic']
};
```

8. <style scoped>

scoped styles only affect the template of the component in which you define them

9. Dynamic Components

```
<div>
  <the-header></the-header>
  <button @click="setSelectedComponent('active-goals')">Active Goals</button>
  <button @click="setSelectedComponent('manage-goals')">Manage Goals</button>
  <!-- <active-goals v-if="selectedComponent === 'active-goals'"></active-goals>
  <manage-goals v-if="selectedComponent === 'manage-goals'"></manage-goals> -->
  <component :is="selectedComponent"></component>
</div>
```

keep dynamic components alive:

the state will be saved, like the value of input

```
<keep-alive>
  <component :is="selectedComponent"></component>
</keep-alive>
```

10. Teleport

```
<div>
  <h2>Manage Goals</h2>
  <input type="text">
  <button>Set Goal</button>
  <dialog open data-v-2574a738>…</dialog> == $0
</div>
```

the error-alert is treated as overlay to the entire page, so teleport allows to render the content in the element that we define, here the body element.

```
<teleport to="body">
  <error-alert v-if="inputIsInvalid">
    <h2>Input is invalid!</h2>
    <p>Please enter at least a few characters...</p>
    <button @click="confirmError">Okay</button>
  </error-alert>
</teleport>
```

```
<body>
  <noscript>…</noscript>
  <div id="app" data-v-app>…</div>
  <!-- built files will be auto injected -->
  <script type="text/javascript" src="/js/chunk-vendors.js">
  </script>
  <script type="text/javascript" src="/js/app.js"></script>
  <dialog open data-v-2574a738>…</dialog> == $0
</body>
```

11. v-model

11.1. number

```
<input id="age" name="age" type="number" v-model.number="userAge" ref="ageInput"/>
```

the result of userAge will stay as a number

11.2. checkbox (multiple)

```
<div>
  <input id="interest-news" name="interest" type="checkbox" value="news"
  v-model="interest"/>
  <label for="interest-news">News</label>
</div>
```

11.3. checkbox (single)

```
<div class="form-control">
  <input type="checkbox" id="confirm-terms" name="conform-terms" v-model="confirm"/>
  <label for="confirm-terms">Agree to terms of use?</label>
</div>
```

11.4. radiobox

```
<div>
  <input id="how-video" name="how" type="radio" value="video" v-model="how"/>
  <label for="how-video">Video Courses</label>
</div>
```

```
data() {
  return {
    userName: '',
    userAge: null,
    referrer:'wom',
    interest: [],
    how: null,
    confirm: false
  }
},
```

11.5. v-model in custom component

parent component

```
<div class="form-control">
  <rating-control v-model="rating"></rating-control>
</div>
```

custom component

```
export default {
    props: ['modelValue'],
    emits: ['update:modelValue'],
    methods: {
        activate(option) {
            this.$emit('update:modelValue', option);
        }
    },
}
```

12. Http request

12.1. Firebase – Realtime database

12.2. Sending request

```
fetch(
  'https://vue-http-demo-b6f4c-default-rtdb.firebaseio.com/surveys.json',
  {
    method: 'POST',
    headers: {
      'Content-Type': 'application/json'
    },
    body: JSON.stringify({
      name: this.enteredName,
      rating: this.chosenRating
    })
  }
).then((response) => {
  if(response.ok){
    //
  } else {
    throw new Error('Could not save data!');
  }
}).catch((err) => {
  console.log(err);
  this.error = err.message;
});
```

12.3. Getting data

```
loadExperience(){
  this.isLoading = true;
  fetch('https://vue-http-demo-b6f4c-default-rtdb.firebaseio.com/surveys.
  json')
  .then((response) => {
    if(response.ok) {
      return response.json();
    }
  })
  .then((data) => {
    this.isLoading =false;
    const results = [];
    for(const id in data) {
      results.push({
        id: id,
        name: data[id].name,
        rating: data[id].rating
      });
    }
    this.results = results;
  })
  .catch((err) => {
    this.isLoading =false;
    console.log(err);
    this.error = 'Failed to fetch data - please try again later.';
  });
}
```

```
async loadCoaches(context) {
    const response = await fetch('https://vue-main-3be4f-default-rtdb.firebaseio.com/
    coaches.json');

    if(!response.ok) {
        console.log("not ok");
    }
    const responseData = await response.json();

    const coaches = [];
    for (const key in responseData) {
        const coach = {
            id: key,
            firstName: responseData[key].firstName,
            lastName: responseData[key].lastName,
            description: responseData[key].description,
            hourlyRate: responseData[key].hourlyRate,
            areas: responseData[key].areas,
        };
        coaches.push(coach);
    }
    context.commit('setCoaches', coaches);
}
```

12.4. Load data automatically (mount)

```
mounted() {
  this.loadExperience();
}
```

13. Routing

13.1. setup

npm install vue-router@next --save

```
import { createApp } from 'vue';
import { createRouter, createWebHistory } from 'vue-router';
import App from './App.vue';
import TeamsList from './components/teams/TeamsList.vue';
import UsersList from './components/users/UsersList.vue';

const router = createRouter({
    history: createWebHistory(),
    routes: [
        {
            path: '/teams',
            component: TeamsList
        },
        {
            path: '/users',
            component: UsersList
        },
    ],
    linkActiveClass: 'active'
});
const app = createApp(App);

app.use(router);

app.mount('#app');
```

13.2. main.js

```
<main>
  <router-view></router-view>
</main>
```

13.3. A

13.4. ro

```
<nav>
  <ul>
    <li>
      <router-link to="/teams">Teams</router-link>
    </li>
    <li>
      <router-link to="/users">Users</router-link>
    </li>
  </ul>
</nav>
```

13.5. pr

```
methods: {
  confirmInput() {
    this.$router.push('/teams');
  }
},
```

13.6. Passing data with Route Params

---- main.js

```
const router = createRouter({
  history: createWebHistory(),
  routes: [
    {
      path: '/teams',
      component: TeamsList
    },
    {
      path: '/users',
      component: UsersList
    },
    {
      path: '/teams/:teamId',
      component: TeamMembers
    }
  ],
  linkActiveClass: 'active'
});
```

—— TeamMembers.js (inject data)

```
data() {
  return {
    teamName: '',
    members: []
  };
},
created() {
  //this.$route.path // /teams/t1
  const teamId = this.$route.params.teamId;
  const selectedTeam = this.teams.find(team => team.id === teamId);

  const members = selectedTeam.members;
  const selectedMembers = [];
  for (const member of members) {
      const selectedUser = this.users.find(user => user.id === member);
      selectedMembers.push(selectedUser);
  }
  this.members = selectedMembers;
  this.teamName = selectedTeam.name;

},
```

13.7. Dynamic path

```
<li>
  <h3>{{ name }}</h3>
  <div class="team-members">{{ memberCount }} Members</div>
  <router-link :to="'/teams/' + id">View Members</router-link>
</li>
```

13.8. Update params with watcher

```
watch: {
  $route(newValue) {
    this.loadTeamMembers(newValue);
  }
}
```

13.9. Passing params as props

main.js

```
{
  path: '/teams/:teamId',
  component: TeamMembers,
  props: true
}
```

dynamic parameters will be passed into this component as props

TeamMembers.vue

```
export default {
  inject: ['users', 'teams'],
  props: ['teamId'],
  components: {
    UserItem
  },
```

13.10. Redirecting & catch all

```
{
    path: '/',
    redirect: '/teams'
},
{
  path: '/teams',
  component: TeamsList,
  alias: '/'
},
```

- the home page redirect to the TeamsList component, and url changes
- alias do the same thing but url don't change

```
{
    path: '/:notFound(.*)',
    component: NotFound
}
```

13.11. Nested Routes

```
{
  path: '/teams',
  component: TeamsList,
  children: [
    {
      path: ':teamId',
      component: TeamMembers,
      props: true
    }
  ]
},
```

```
<template>
  <router-view></router-view>
  <ul>
    <teams-item
      v-for="team in teams"
      :key="team.id"
      :id="team.id"
      :name="team.name"
      :member-count="team.members.length"
    ></teams-item>
  </ul>
</template>
```

It allows the child component display with the parent on the same page, don't forget to add a router-view in the parent component

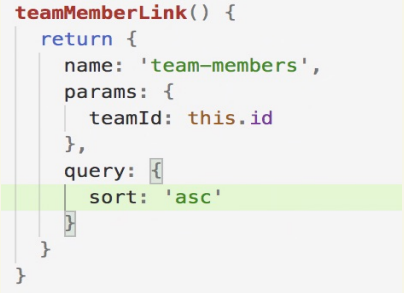
13.12. Named Route

```
{
  name: 'teams',
  path: '/teams',
  component: TeamsList,
  children: [
    {
      name: 'team-members',
      path: ':teamId',
      component: TeamMembers,
      props: true
    }
  ]
},
```

```
computed: {
  teamMemberLink() {
    return {
      name: 'team-members',
      params: {
        teamId: this.id
      }
    }
  }
},
```

we don't need to update the path if it is changed

13.13. Query params



```
teamMemberLink() {
  return {
    name: 'team-members',
    params: {
      teamId: this.id
    },
    query: {
      sort: 'asc'
    }
  }
}
```

13.14. Multiple routes with named router views

```
{
  name: 'teams',
  path: '/teams',
  components: { default: TeamsList, footer: TeamsFooter },
  children: [
    {
      name: 'team-members',
      path: ':teamId',
      component: TeamMembers,
      props: true
    }
  ]
},
```

```
<template>
  <the-navigation></the-navigation>
  <main>
    <router-view></router-view>
  </main>
  <footer>
    <router-view name="footer"></router-view>
  </footer>
</template>
```

13.15. Scroll behavior

```
scrollBehavior(to, from, savedPosition) {
    console.log(to, from, savedPosition);
    if(savedPosition) {
        return savedPosition;
    }
    return { left: 0, top: 0 };
}
```

13.16. Navigation guards

```
router.beforeEach(function(to, from, next) {
    console.log(to, from);
    next(false);
});
```

Before each navigation, this function decides if the navigation is allowed or not. next( ), next(false), next({ A view } } are possible. This is global.

For single route, we have **beforeEnter** in main.js or **beforeRouteEnter** in component.

```
beforeRouteLeave (to, from, next) {
  console.log('Before Route Leave');
  console.log(to, from);
  if(this.changeSaved) {
    next();
  } else {
    const userWantToLeave = confirm('Are you sure? You got unsaved changes.');
    next(userWantToLeave);
  }
}
```

Make sure users really want to leave the page even they have unsaved changes.

14. Vuex

14.1. setup

npm install --save vuex@next

14.2. global state

main.js

```
import { createApp } from 'vue';
import { createStore } from 'vuex';

import App from './App.vue';

const store = createStore({
    state() {
        return {
            counter: 0
        };
    }
});
const app = createApp(App);

app.use(store);
app.mount('#app');
```

App.vue

```
<template>
  <base-container title="Vuex">
  <h3>{{ $store.state.counter }}</h3>
  <button>Add 1</button>
  </base-container>
</template>
```

14.3. global Methode for state — mutations

main.js

```
const store = createStore({
    state() {
        return {
            counter: 0
        };
    },
    mutations: {
        increment(state) {
            state.counter++;
        }
    }
});
```

App.vue

```
methods: {
  addOne() {
    this.$store.commit('increment');
  }
},
```

14.4. Passing data to Mutations with payload

```
mutations: {
    increment(state) {
        state.counter++;
    },
    increase(state, payload) {
        state.counter = state.counter + payload.value;
    }
}
```

```
methods: {
  addOne() {
    this.$store.commit('increase', { value: 10 });
  }
},
```

payload can be a number, a string , or an object.

14.5. Getters

If we have two same functions in different components, maybe these two functions execute at the same time, which is not we want. So we use Getters to get rid of that.

main.js

```
getters: {
    finalCounter(state) {
        return state.counter * 3;
    },
    normalizedCounter(_, getters) {
        const finalCounter = getters.finalCounter;
        if(finalCounter < 0) {
            return 0;
        }
        if(finalCounter > 100) {
            return 100;
        }
        return finalCounter;
    }
}
```

In Component

```
computed: {
  counter() {
    return this.$store.getters.normalizedCounter;
  }
}
```

14.6. Actions

mutation is synchronous (即时， 同步)，but sometimes we need to wait until the action finish, which is asynchronous(异步，延迟), so we should use actions.

main.js

```
actions: {
    increment(context) {
        setTimeout(function(){
            context.commit('increment');
        }, 2000);
    },
},
```

In component

```
methods: {
  addOne() {
    this.$store.dispatch('increment');
  }
}
```

14.7. Mapper

mapGetters gives an object full of computed properties, so we don't have to write computed functions ourselves.

In component

```
<template>
  <h3>{{ finalCounter }}</h3>
</template>

<script>
import { mapGetters } from 'vuex';
export default {
  computed: {
    // counter() {
    //   return this.$store.getters.finalCounter;
    // }
    ...mapGetters(['finalCounter'])
  }
};
</script>
```

mapActions provides all the actions we registered.

```
<template>
  <button @click="increment">Add 1</button>
</template>

<script>
import { mapActions } from 'vuex';
export default {
  methods: {
    // addOne() {
    //   this.$store.dispatch('increment');
    // }
    ...mapActions(['increment'])
  }
};
</script>
```

14.8. Modules

```
const store = createStore({
    modules: {
        prods: productModule,
        cart: cartModule
    },
```

```
const productModule = {
  namespaced: true,
  state() {
    return {
      products: [
```

——Getters

```
cart() {
  return this.$store.getters['cart/getCart'];
}
```

——Actions

```
methods: {
  remove() {
    this.$store.dispatch('cart/removeFromCart', this.prodId);
  }
}
```

——Mappers

```
computed: {
  ...mapGetters({
    isLoggedIn: 'userIsAuthenticated',
    cartQty: 'cart/quantity'
  }),

},
```

Main Projet

1. npm install --save vue-router@next vuex@next
2. Modal window transition

BaseDialog.vue

```vue
<template>
  <teleport to="body">
    <div v-if="show" @click="tryClose" class="backdrop"></div>
    <transition name="dialog">
      <dialog open v-if="show">
        <header>
          <slot name="header">
            <h2>{{ title }}</h2>
          </slot>
        </header>
        <section>
          <slot></slot>
        </section>
        <menu v-if="!fixed">
          <slot name="actions">
            <base-button @click="tryClose">Close</base-button>
          </slot>
        </menu>
      </dialog>
    </transition>
  </teleport>
</template>
```

CSS

```css
.dialog-enter-from,
.dialog-leave-to {
  opacity: 0;
  transform: scale(0.8);

}

.dialog-enter-active {
  transition: all 0.3s ease-out;
}

.dialog-leave-active {
  transition: all 0.3s ease-in;
}

.dialog-enter-to,
.dialog-leave-from {
  opacity: 1;
  transform: scale(1);

}
```

3. window transition

App.vue

```vue
<template>
  <the-header></the-header>
  <router-view v-slot="slotProps">
    <transition name="route" mode="out-in">
      <component :is="slotProps.Component"></component>
    </transition>
  </router-view>
</template>
```

CSS

```css
.route-enter-from {
  opacity: 0;
  transform: translateY(-30px);
}

.route-leave-to {
  opacity: 0;
  transform: translateY(30px);
}

.route-enter-to ,
.route-leave-from {
  opacity: 1;
  transform: translateY(0);
}

.route-enter-active {
  transition: all 0.3s ease-out;
}

.route-leave-active {
  transition: all 0.3s ease-in;
}
```

4. Authentication
   4.1. firebase rules

```
{
  "rules": {
    "coaches": {
      ".read": true,
      ".write": "auth != null"
    },
    "requests": {
      ".read": "auth != null",
      ".write": true
    },
  }
}
```

4.2. Output data when with is necessary

```
async loadRequests(context) {
    const coachId = context.rootGetters.userId;
    const token = context.rootGetters.token;
    const response = await fetch(`https://vue-main-3be4f-default-rtdb.firebaseio.com/
    requests/${coachId}.json?auth=` + token);
    const responseData = await response.json();
    if(!response.ok) {
        const err = new Error(responseData.message || 'Failed to fetch!');
        throw err;
    }
    const requests = [];
    for(const key in responseData) {
        const request = {
            id:key,
            coachId: coachId,
            userEmail: responseData[key].userEmail,
            message: responseData[key].message
        };
        requests.push(request);
    }
    context.commit('setRequests', requests);
}
```

5. Optimize ——don't download all components (using Asynchronous)

main.js

```
import { createApp, defineAsyncComponent } from 'vue';…
// import BaseDialog from './components/ui/BaseDialog.vue';

const BaseDialog = defineAsyncComponent(() => import('./components/ui/BaseDialog.vue'));
const app = createApp(App);

app.use(router);
app.use(store);
app.component('base-card', BaseCard);
app.component('base-button', BaseButton);
app.component('base-badge', BaseBadge);
app.component('base-spinner', BaseSpinner);
app.component('base-dialog', BaseDialog);

app.mount('#app');
```

router.js

```
const CoachDetail = () =>
  import('./pages/coaches/CoachesList.vue');
```

6. Deploy App
    6.1. npm install -g firebase-tools
    6.2. firebase login
    6.3. firebase init
    6.4. firebase deploy

Composition API

1. setup( ) = data + methods + computed + watch
2. data( )
    - ref works with string, number or object, but we should manipulate with ref.value.

```
import { ref } from 'vue';
export default {
  setup() {
    const user = ref({
      name: 'Maximilian',
      age: 31
    });
    setTimeout(function() {
      user.value.name = 'Max';
      user.value.age = 32
    }, 2000);
    return { user: user };
  }
};
```

- reactive only works with object, but we can use the object directly.

```
<script>
import { reactive } from 'vue';
export default {
  setup() {
    const user = reactive({
      name: 'Maximilian',
      age: 31
    });
    setTimeout(function() {
      user.name = 'Max';
      user.age = 32
    }, 2000);
    return { user: user };
  }
};
</script>
```

3. methods ——function in setup( )

```
function setNewAge() {
  user.age = 32;
}
```

4. computed

```
import { ref, computed } from 'vue';
export default {
  setup() {
    const firstName = ref('');
    const lastName = ref('');
    const uAge = ref(31);

    const uName = computed(function() {
      return firstName.value + ' ' + lastName.value;
    });
```

5. Template refs

```
<div>
  <input type="text" placeholder="First Name" v-model="firstName"/>
  <input type="text" placeholder="Last Name" ref="lastNameInput"/>
  <button @click="setLastName">set LastName</button>
</div>
```

```
const lastNameInput = ref(null);

function setLastName() {
  lastName.value = lastNameInput.value.value;
}
```

6. Components, props & composition API

In component

```
import { computed } from 'vue';
export default {
    props:['firstName', 'lastName', 'age'],
    setup(props) {
        const uName = computed(function() {
            return props.firstName + ' ' + props.lastName;
        });

        return { userName: uName };
    }
}
```

7. Custom Events

In Component

```
<template>
    <h2>{{ userName }}</h2>
    <h3>{{ age }}</h3>
    <div>
    <input type="text" placeholder="Favorite color" v-model="f_color"/>
    <button @click="saveData">Set favorite color</button>
    </div>
</template>
```

```
setup(props, context) {
    const uName = computed(function() {
        return props.firstName + ' ' + props.lastName;
    });
    const f_color = ref('');
    function saveData() {
        context.emit('save-data', f_color);
    }
    return { userName: uName, f_color, saveData };
}
```

App.vue

```
<user-data
  :first-name="firstName"
  :last-name="lastName"
  :age="age"
  @save-data="saveData"
></user-data>
```

```
const color = ref('');
function saveData(f_color) {
  color.value = f_color.value;
}
```

8. Provide & Inject

App.vue

```
const uAge = ref(31);
provide('userAge', uAge);
```

In Component

```
const age = inject('userAge');
```

9. Lifecycle Hooks

```
import {
  ref,
  computed,
  inject,
  onBeforeMount,
  onMounted,
  onBeforeUpdate,
  onUpdated,
  onBeforeUnmount,
  onUnmounted
} from 'vue';
```

10. Route

```
import { useRoute } from 'vue-router';

const route = useRoute();
console.log(route.params.pid);
```

11. Router

```
import { useRouter } from 'vue-router';

const router = useRouter();
router.push('/products');
```

12. Vuex ——store

```
import { useStore} from 'vuex';

const store =useStore();
store.dispatch('products');
```