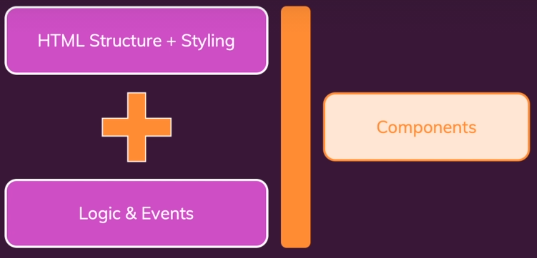
**Section 20 – Reusing Functionality Mixins And**

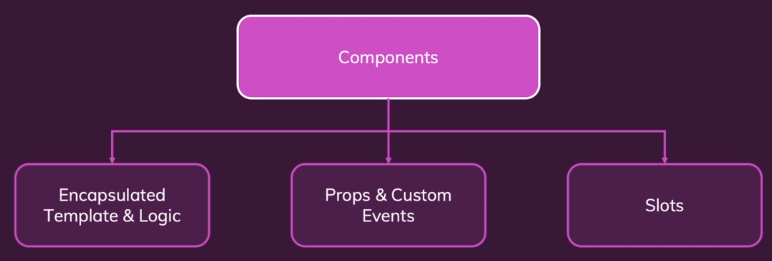
**Custom Composition Functions**

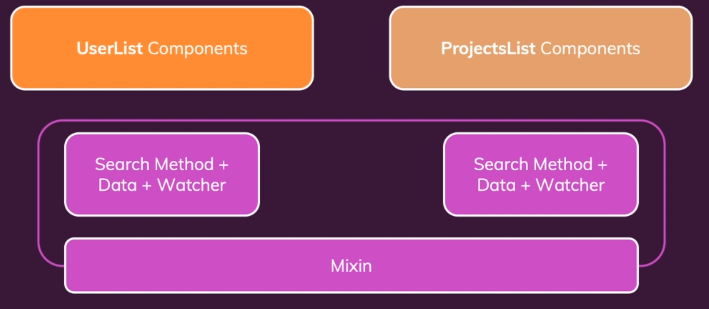
Chapter 303 – Reusability Concepts

1. What can we reuse ?



1. We have add user and delete user which has same structure in the alert.





Chapter 304 – Using Mixins

1. Use Mixins in AddUser.vue and DeleteUser.vue

<script>

import UserAlert from './UserAlert.vue';

import alertMixin from '../mixins/alert.js';

export default {

  components: {

      UserAlert,

  },

  mixins: [alertMixin],

};

</script>

In alert.js

export default {

    data() {

      return {

        alertIsVisible: false,

      };

    },

    methods: {

      showAlert() {

        this.alertIsVisible = true;

      },

      hideAlert() {

        this.alertIsVisible = false;

      },

    },

};

Chapter 305 – Understanding Mixin Merging

1. If we have two data, one in DeleteUser.vue one in alert.js which is the mixins file, Vue will read both of it.

<script>

import UserAlert from './UserAlert.vue';

import alertMixin from '../mixins/alert.js';

export default {

  components: {

      UserAlert,

  },

  data() {

    return {

      alertTitle: 'Delete User ?'

    };

  },

  mixins: [alertMixin],

};

</script>

export default {

    data() {

      return {

        alertIsVisible: false,

      };

    },

    methods: {

      showAlert() {

        this.alertIsVisible = true;

      },

      hideAlert() {

        this.alertIsVisible = false;

      },

    },

};

But if there are two function same between alert.js and DeleteUser.vue, the function in DeleteUser.vue will be selected, and the function in alert.js will not be selected.

Chapter 306 – Global Mixins

1. We can add global mixins by adding mixins file such as logger.js and import it into main.js. It can be called every things if it is mounted.

export default {

    mounted() {

        console.log('Mounted');

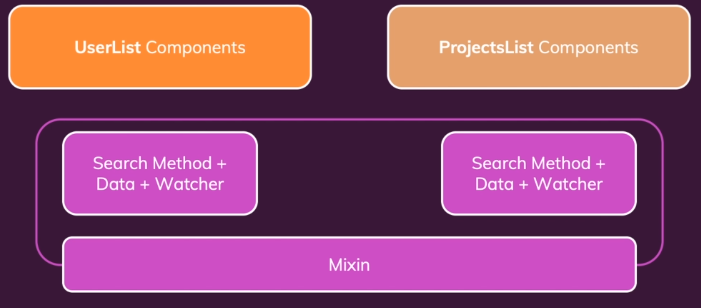
    }

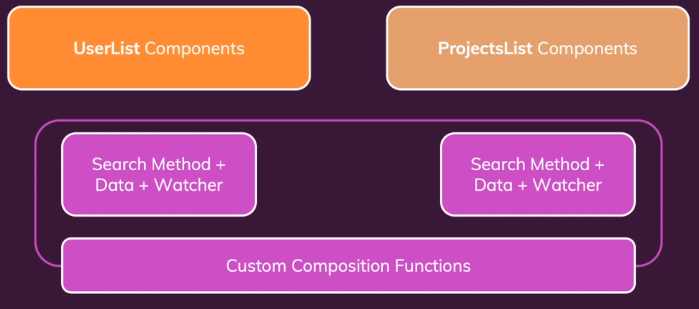
};

Chapter 307 – Disadvantages of Mixins

1. Hard to manage in big project because we are not sure where the code come from.







Chapter 308 – Custom Hooks / Composables And The Composition API

1. We create custom hooks and start the function using “use”. We can use ref [] to get the returned data. Because it is a ref and we’re exposing this ref to the template and whenever you do that Vue is aware of it and whenever data ref changes, these changes are reflected in the template. So it is a different mental model, but in the end it just a function which we defined on our own which we are calling here.

Alert.js

import { ref } from 'vue';

export default function useAlert() {

    const alertIsVisible = ref(false);

    function showAlert() {

      alertIsVisible.value = true;

    }

    function hideAlert() {

      alertIsVisible.value = false;

    }

    return [

      alertIsVisible,

      showAlert,

      hideAlert

    ];

}

AddUser.vue

<script>

import UserAlert from './UserAlert.vue';

import useAlert from '../hooks/alert.js';

export default {

  components: {

    UserAlert,

  },

  setup() {

    const [alertIsVisible, showAlert, hideAlert] = useAlert();

    return {

      alertIsVisible,

      showAlert,

      hideAlert

    }

  },

};

</script>

Chapter 309 – More Custom Composition Functions

1. We can send a parameter into our hooks.

import { ref } from 'vue';

export default function useAlert(startingVisibility = false) {

    // const alertIsVisible = ref(false);

    const alertIsVisible = ref(startingVisibility);

    function showAlert() {

      alertIsVisible.value = true;

    }

    function hideAlert() {

      alertIsVisible.value = false;

    }

    return [

      alertIsVisible,

      showAlert,

      hideAlert

    ];

}

<script>

import { ref } from 'vue';

import UserAlert from './UserAlert.vue';

import useAlert from '../hooks/alert.js';

export default {

  components: {

    UserAlert,

  },

  setup() {

    const alertTitle = ref('Delete User?');

    const [alertIsVisible, showAlert, hideAlert] = useAlert(true);

    return {

      alertIsVisible,

      showAlert,

      hideAlert,

      alertTitle

    }

  },

};

</script>

Chapter 310 – Why Hooks / Composables Beat Mixins

1. In Mixins we are not sure which mixins is used, but in hooks wh know the things that we wants to expose to a component (return value) and therefore, we have to accept these things like here, where we get an array and pull the things we need out of the array and store it on the constant.
2. It is super easy to see that alertIsVisible, showAlert, and hideAlert are coming from the useAlert hook. Event we use 10 or 20 custom hoks, it would always be easy to know which data piece come from which hook.

Chapter 311 – Example: Creating A “Search” Hook

1. Props will not exist in search.js or hooks file, event we called it in UserList.vue or original file. Therefore, we need to add a parameter and maybe name it items. We also add one parameter for filteredItems (filtered index).

Search.js

import { ref, computed, watch } from 'vue';

export default function useSearch(items, searchProp) {

    const enteredSearchTerm = ref('');

    const activeSearchTerm = ref('');

    // const availableUsers = computed(function () {

    const availableItems = computed(function () {

    //   let users = [];

    let filteredItems = [];

      if (activeSearchTerm.value) {

        // users = props.users.filter((usr) =>

        filteredItems = items.filter((item) =>

            //   usr.fullName.includes(activeSearchTerm.value)

            item[searchProp].includes(activeSearchTerm.value)

        );

        // The code above only works for user because

        // it is only filtered using fullName

      // } else if (props.users) {

      } else if (items) {

        // users = props.users;

        filteredItems = items;

      }

    //   return users;

      return filteredItems;

    });

    watch(enteredSearchTerm, function (newValue) {

      setTimeout(() => {

        if (newValue === enteredSearchTerm.value) {

          activeSearchTerm.value = newValue;

        }

      }, 300);

    });

    function updateSearch(val) {

      enteredSearchTerm.value = val;

    }

    return {

        enteredSearchTerm,

        availableItems,

        updateSearch

    };

}

UserList.vue

<script>

// import { ref, computed, watch } from 'vue';

import { ref, computed } from 'vue';

import UserItem from './UserItem.vue';

import useSearch from '../../hooks/search.js';

export default {

  components: {

    UserItem,

  },

  props: ['users'],

  emits: ['list-projects'],

  setup(props) {

    // const sorting = ref(null);

    // const displayedUsers = computed(function () {

    //   if (!sorting.value) {

    //     return availableUsers.value;

    //   }

    //   return availableUsers.value.slice().sort((u1, u2) => {

    //     if (sorting.value === 'asc' && u1.fullName > u2.fullName) {

    //       return 1;

    //     } else if (sorting.value === 'asc') {

    //       return -1;

    //     } else if (sorting.value === 'desc' && u1.fullName > u2.fullName) {

    //       return -1;

    //     } else {

    //       return 1;

    //     }

    //   });

    // });

    // function sort(mode) {

    //   sorting.value = mode;

    // }

    const { enteredSearchTerm, availableItems, updateSearch } = useSearch(

      props.users,

      'fullName'

    );

    const sorting = ref(null);

    const displayedUsers = computed(function () {

      if (!sorting.value) {

        return availableItems.value;

      }

      return availableItems.value.slice().sort((u1, u2) => {

        if (sorting.value === 'asc' && u1.fullName > u2.fullName) {

          return 1;

        } else if (sorting.value === 'asc') {

          return -1;

        } else if (sorting.value === 'desc' && u1.fullName > u2.fullName) {

          return -1;

        } else {

          return 1;

        }

      });

    });

    function sort(mode) {

      sorting.value = mode;

    }

    return {

      enteredSearchTerm,

      updateSearch,

      displayedUsers,

      sorting,

      sort

    };

  },

  // data() {

  //   return {

  //     enteredSearchTerm: '',

  //     activeSearchTerm: '',

  //     sorting: null,

  //   };

  // },

  // computed: {

  //   availableUsers() {

  //     let users = [];

  //     if (this.activeSearchTerm) {

  //       users = this.users.filter((usr) =>

  //         usr.fullName.includes(this.activeSearchTerm)

  //       );

  //     } else if (this.users) {

  //       users = this.users;

  //     }

  //     return users;

  //   },

  //   displayedUsers() {

  //     if (!this.sorting) {

  //       return this.availableUsers;

  //     }

  //     return this.availableUsers.slice().sort((u1, u2) => {

  //       if (this.sorting === 'asc' && u1.fullName > u2.fullName) {

  //         return 1;

  //       } else if (this.sorting === 'asc') {

  //         return -1;

  //       } else if (this.sorting === 'desc' && u1.fullName > u2.fullName) {

  //         return -1;

  //       } else {

  //         return 1;

  //       }

  //     });

  //   },

  // },

  // methods: {

  //   updateSearch(val) {

  //     this.enteredSearchTerm = val;

  //   },

  //   sort(mode) {

  //     this.sorting = mode;

  //   },

  // },

  // watch: {

  //   enteredSearchTerm(val) {

  //     setTimeout(() => {

  //       if (val === this.enteredSearchTerm) {

  //         this.activeSearchTerm = val;

  //       }

  //     }, 300);

  //   }

  // },

};

</script>

Chapter 312 – Custom Hooks Gotchas

1. We can use the same search.js for ProjectList, just change the parameter.

ProjectList.vue

<script>

import { computed, watch, toRefs } from 'vue';

import ProjectItem from './ProjectItem.vue';

import useSearch from '../../hooks/search.js';

export default {

  components: {

    ProjectItem,

  },

  props: ['user'],

  setup(props) {

    const { user } = toRefs(props);

    const projects = computed(function () {

      return user.value ? user.value.projects : [];

    });

    // const enteredSearchTerm = ref('');

    // const activeSearchTerm = ref('');

    // const availableProjects = computed(function () {

    //   if (activeSearchTerm.value) {

    //     return props.user.projects.filter((prj) =>

    //       prj.title.includes(activeSearchTerm.value)

    //     );

    //   }

    //   return props.user.projects;

    // });

    const { enteredSearchTerm, availableItems, updateSearch } = useSearch(

      projects, // ref

      'title'

    );

    const hasProjects = computed(function () {

      // return props.user.projects && availableProjects.value.length > 0;

      return props.user.projects && availableItems.value.length > 0;

    });

    // watch(enteredSearchTerm, function (newValue) {

    //   setTimeout(() => {

    //     if (newValue === enteredSearchTerm.value) {

    //       activeSearchTerm.value = newValue;

    //     }

    //   }, 300);

    // });

    // const propsWithRefs = toRefs(props);

    // const user = propsWithRefs.user;

    watch(user, function () {

      enteredSearchTerm.value = '';

    });

    // function updateSearch(val) {

    //   enteredSearchTerm.value = val;

    // }

    return {

      enteredSearchTerm,

      availableProjects: availableItems,

      hasProjects,

      updateSearch,

    };

  },

  // data() {

  //   return {

  //     enteredSearchTerm: '',

  //     activeSearchTerm: '',

  //   };

  // },

  // computed: {

  //   hasProjects() {

  //     return this.user.projects && this.availableProjects.length > 0;

  //   },

  //   availableProjects() {

  //     if (this.activeSearchTerm) {

  //       return this.user.projects.filter((prj) =>

  //         prj.title.includes(this.activeSearchTerm)

  //       );

  //     }

  //     return this.user.projects;

  //   },

  // },

  // methods: {

  //   updateSearch(val) {

  //     this.enteredSearchTerm = val;

  //   },

  // },

  // watch: {

  //   enteredSearchTerm(val) {

  //     setTimeout(() => {

  //       if (val === this.enteredSearchTerm) {

  //         this.activeSearchTerm = val;

  //       }

  //     }, 300);

  //   },

  //   user() {

  //     this.enteredSearchTerm = '';

  //   },

  // },

};

</script>

Changes of search.js

import { ref, computed, watch } from 'vue';

export default function useSearch(items, searchProp) {

    const enteredSearchTerm = ref('');

    const activeSearchTerm = ref('');

    // const availableUsers = computed(function () {

    const availableItems = computed(function () {

    //   let users = [];

    let filteredItems = [];

      if (activeSearchTerm.value) {

        // users = props.users.filter((usr) =>

        filteredItems = items.value.filter((item) =>

            //   usr.fullName.includes(activeSearchTerm.value)

            item[searchProp].includes(activeSearchTerm.value)

        );

        // The code above only works for user because

        // it is only filtered using fullName

      // } else if (props.users) {

      } else if (items.value) {

        // users = props.users;

        filteredItems = items.value;

      }

    //   return users;

      return filteredItems;

    });

    watch(enteredSearchTerm, function (newValue) {

      setTimeout(() => {

        if (newValue === enteredSearchTerm.value) {

          activeSearchTerm.value = newValue;

        }

      }, 300);

    });

    function updateSearch(val) {

      enteredSearchTerm.value = val;

    }

    return {

        enteredSearchTerm,

        availableItems,

        updateSearch

    };

}

Chapter 313 – More Thoughts On Custom Hooks / Composables

1. We can use function to update the value like below.

<script>

import { computed, watch, toRefs } from 'vue';

import ProjectItem from './ProjectItem.vue';

import useSearch from '../../hooks/search.js';

export default {

  components: {

    ProjectItem,

  },

  props: ['user'],

  setup(props) {

    const { user } = toRefs(props);

    const projects = computed(function () {

      return user.value ? user.value.projects : [];

    });

    const { enteredSearchTerm, availableItems, updateSearch } = useSearch(

      projects, // ref

      'title'

    );

    const hasProjects = computed(function () {

      // return props.user.projects && availableProjects.value.length > 0;

      return user.value.projects && availableItems.value.length > 0;

    });

    watch(user, function () {

      // enteredSearchTerm.value = '';

      updateSearch('');

    });

    return {

      enteredSearchTerm,

      availableProjects: availableItems,

      hasProjects,

      updateSearch,

    };

  },

};

</script>

Chapter 314 – Example: A Custom “Sort” Hook

1. We can also make a hooks file for sorting.

Sort.js

import { ref, computed } from 'vue';

export default function useSort(availableItems, sortProperty) {

    const sorting = ref(null);

    const displayedUsers = computed(function () {

      if (!sorting.value) {

        return availableItems.value;

      }

      return availableItems.value.slice().sort((u1, u2) => {

        if (sorting.value === 'asc' && u1[sortProperty] > u2[sortProperty]) {

          return 1;

        } else if (sorting.value === 'asc') {

          return -1;

        } else if (sorting.value === 'desc' && u1[sortProperty] > u2[sortProperty]) {

          return -1;

        } else {

          return 1;

        }

      });

    });

    function sort(mode) {

      sorting.value = mode;

    }

    return {

        sorting,

        displayedUsers,

        sort

    }

}

UserList.vue

<script>

// import { ref, computed, watch } from 'vue';

import { toRefs } from 'vue';

import UserItem from './UserItem.vue';

import useSearch from '../../hooks/search.js';

import useSort from '../../hooks/sort.js';

export default {

  components: {

    UserItem,

  },

  props: ['users'],

  emits: ['list-projects'],

  setup(props) {

    const { users } = toRefs(props);

    const { enteredSearchTerm, availableItems, updateSearch } = useSearch(

      users, // not ref

      'fullName'

    );

    const { sorting, displayedUsers, sort } = useSort(

      availableItems,

      'fullName'

    );

    return {

      enteredSearchTerm,

      updateSearch,

      displayedUsers,

      sorting,

      sort

    };

  },

};

</script>