**Section 13 – Routing Building Multi Page In Single Page Applications**

Chapter 163 – What And Why ?

1. Routing is used to fetch one single HTML file and that file includes import to the script with JavaScript code which controlling it with the same URL.

Chapter 164 – Routing Setup

1. To install router using this code

npm install --save vue-router

Then, restart your app using

npm run serve

1. Setup

import { createRouter } from 'vue-router';

const router = createRouter({

    history: createRouterHistory(),

    routes: []

});

Chapter 165 – Registering And Rendering Routes

1. This app.use() below allows us to connect our Vue app, with a third party package.

app.use(router);

1. This code below tells us the place routed to component and should be loaded.

<router-view></router-view>

Now you can navigate to two different page HTML with different URL, but we don’t want to access it by URL, you need to make it easily by pressing the navigation.

Chapter 166 – Navigating With Router Link

1. Because we will use router, so we don’t need to navigate the HTML page one by one. We can comment this code below.

export default {

  emits: ['set-page'],

  methods: {

    setActivePage(page) {

      this.$emit('set-page', page);

    },

  },

};

We also don’t need this.

<button @click="setActivePage('teams-list')">Teams</button>

Chapter 167 – Styling Active Links

1. Add styling to the Active Class

const router = createRouter({

  history: createWebHistory(),

  routes: [

    { path: '/teams', component: TeamsList }, // our-domain.com/teams => TeamsList

    { path: '/users', component: UsersList },

  ],

  linkActiveClass: 'active'

});

Chapter 168 – Programmatic Navigation

1. Navigate to the other link using button

export default {

  components: {

    UserItem,

  },

  inject: ['users'],

  methods: {

    confirmInput() {

      this.$router.push('/teams');

    }

  }

};

Chapter 169 – Passing Data With Route Params (Dynamic Segments)

1. Do Dynamic Routing to access anything link after /teams/…

const router = createRouter({

  history: createWebHistory(),

  routes: [

    { path: '/teams', component: TeamsList }, // our-domain.com/teams => TeamsList

    { path: '/users', component: UsersList },

    { path: '/teams/:teamId', component: TeamMembers },

  ],

  linkActiveClass: 'active'

});

And to get the data we can use this code in TeamMember.vue

export default {

  inject: ['users', 'teams'],

  components: {

    UserItem

  },

  data() {

    return {

      teamName: '',

      members: []

      // teamName: 'Test',

      // members: [

      //   { id: 'u1', fullName: 'Max Schwarz', role: 'Engineer' },

      //   { id: 'u2', fullName: 'Max Schwarz', role: 'Engineer' },

      // ],

    };

  },

  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.[selectedUser];

    }

    this.members = selectedMembers;

    this.teamName = selectedTeam.name;

  }

};

Chapter 170 – Navigation And Dynamic Paths

Chapter 171 – Updating Params Data With Watchers

1. We want to make our routing dynamic using some inputted value.

export default {

  inject: ['users', 'teams'],

  components: {

    UserItem

  },

  data() {

    return {

      teamName: '',

      members: []

      // teamName: 'Test',

      // members: [

      //   { id: 'u1', fullName: 'Max Schwarz', role: 'Engineer' },

      //   { id: 'u2', fullName: 'Max Schwarz', role: 'Engineer' },

      // ],

    };

  },

  methods: {

    loadTeamMembers(route) {

      const teamId = 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;

    }

  },

  created() {

    // this.$route.path // /teams/t1

    this.loadTeamMembers(this.$route);

  },

  watch: {

    $route(newRoute) {

      this.loadTeamMembers(newRoute);

    },

  }

};

Chapter 172 – Passing Params As Props

1. The code like below.

export default {

  inject: ['users', 'teams'],

  props: ['teamId'],

  components: {

    UserItem,

  },

  data() {

    return {

      teamName: '',

      members: []

      // teamName: 'Test',

      // members: [

      //   { id: 'u1', fullName: 'Max Schwarz', role: 'Engineer' },

      //   { id: 'u2', fullName: 'Max Schwarz', role: 'Engineer' },

      // ],

    };

  },

  methods: {

    loadTeamMembers(teamId) {

      // const teamId = teamId.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;

    }

  },

  created() {

    // this.$route.path // /teams/t1

    this.loadTeamMembers(this.teamId);

  },

  watch: {

    teamId(newRoute) {

      this.loadTeamMembers(newRoute);

    },

  }

};

Chapter 173 – Redirecting And “Catch All” Routes

1. Redirecting some route into another route, like we want to make a default web ‘/’

const router = createRouter({

  history: createWebHistory(),

  routes: [

    { path: '/', component: TeamsList },

    { path: '/teams', component: TeamsList }, // our-domain.com/teams => TeamsList

    { path: '/users', component: UsersList },

    // { path: '/teams/:teamId', component: TeamMembers },

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

  ],

  linkActiveClass: 'active'

});

Or like below, but below your domain will have ‘/teams’, upper your domain will not have ‘/teams’

const router = createRouter({

  history: createWebHistory(),

  routes: [

    { path: '/', redirect: '/teams' },

    { path: '/teams', component: TeamsList }, // our-domain.com/teams => TeamsList

    { path: '/users', component: UsersList },

    // { path: '/teams/:teamId', component: TeamMembers },

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

  ],

  linkActiveClass: 'active'

});

Or like below.

const router = createRouter({

  history: createWebHistory(),

  routes: [

    // { path: '/', redirect: '/teams' },

    { path: '/teams', component: TeamsList, alias: '/' }, // our-domain.com/teams => TeamsList

    { path: '/users', component: UsersList },

    // { path: '/teams/:teamId', component: TeamMembers },

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

  ],

  linkActiveClass: 'active'

});

1. Not Found page

{ path: '/:notFound(.\*)', redirect: '/teams'}

Or make a new HTML page

import NotFound from './components/nav/NotFound.vue';

The path code like below using component.

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

Chapter 174 – Using Nested Routes

1. Making routes in routes, like ‘teams/t1’

routes: [

    { path: '/', redirect: '/teams' },

    { path: '/teams',

      component: TeamsList,

      children: [

        { path: ':teamId', component: TeamMembers, props: true },

      ] }, // our-domain.com/teams => TeamsList

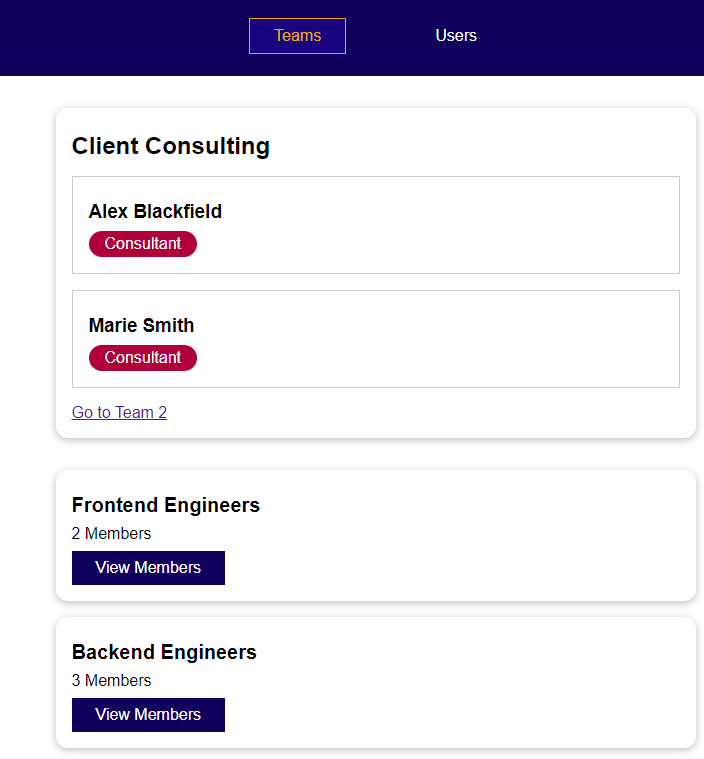
    { path: '/users', component: UsersList },

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

  ],

Chapter 175 – More Fun With Named Routes And Location Objects

1. Add routes result above our menu like the picture below. On the TeamList we may add this.



<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>

Or you can add the route into side note like below.

this.$router.push({ name: 'team-members', params: { teamId: this.Id }});

Chapter 176 – Using Query Params

1. For example we want to sort the result.

export default {

  props: ['id', 'name', 'memberCount'],

  computed: {

    teamMembersLink() {

      // return '/teams/' + this.id + '?sort=asc';

      return {

                name: 'team-members',

                params: { teamId: this.id },

                // path: '/teams' + this.id

                query: { sort: 'asc' }

             };

      // this.$router.push({ name: 'team-members', params: { teamId: this.Id }});

    }

  }

};

Chapter 177 – Rendering Multiple Routes With Named Router Views

1. Using Components instead of Component

routes: [

    { path: '/', redirect: '/teams' },

    {

      name: 'teams',

      path: '/teams',

      components: { default: TeamsList, footer: TeamsFooter },

      children: [

        {

          name: 'team-members',

          path: ':teamId',

          // component: TeamMembers,

          component: TeamMembers,

          props: true },

      ] }, // our-domain.com/teams => TeamsList

    {

      path: '/users',

      components: {

        default: UsersList,

        footer: UsersFooter

      }

    },

    // { path: '/teams/:teamId', component: TeamMembers },

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

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

  ],

  linkActiveClass: 'active'

});

Chapter 178 – Controlling Scroll Behavior

1. Always scroll to Top after clicking some navigation. And we can add saved position so when we want to go back to the previous link, the position will be the same.

const router = createRouter({

  history: createWebHistory(),

  routes: [

    { path: '/', redirect: '/teams' },

    {

      name: 'teams',

      path: '/teams',

      components: { default: TeamsList, footer: TeamsFooter },

      children: [

        {

      ] }, // our-domain.com/teams => TeamsList

    {

      path: '/users',

      components: {

      }

    },

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

  ],

  linkActiveClass: 'active',

  scrollBehavior(to, from, savedPosition) {

    console.log(to, from, savedPosition);

    if (savedPosition) {

      return savedPosition;

    }

    return { left:0,

             top: 0 }; // move to the top of page

  }

});

Chapter 179 – Introducing Navigation Guards

1. Navigate to the link that you have authorization. \_ and \_2 means we have to take these arguments in order to reach the third one, this method like a middleware.

router.beforeEach(function(to, from, next) {

  console.log('Global beforeEach');

  console.log(to, from);

  // next(); // Agree

  // next(false); // Disagree

  if (to.name === 'team-members') {

    next();

  } else {

    next({name: 'team-members', params: { teamId: 't2' }});

  }

});

Chapter 180 – Diving Deeper Into Navigation Guards

1. We can make a new route to check or to do middleware part, like below.

Urutan : Global users, users before enter, userslist cmp before router enter.

const router = createRouter({

  history: createWebHistory(),

  routes: [

    { path: '/', redirect: '/teams' },

    {

      name: 'teams',

      path: '/teams',

      components: { default: TeamsList, footer: TeamsFooter },

      children: [

        {

          name: 'team-members',

          path: ':teamId',

          component: TeamMembers,

          props: true },

      ] }, // our-domain.com/teams => TeamsList

    {

      path: '/users',

      components: {

        default: UsersList,

        footer: UsersFooter

      },

      beforeEnter(to, from, next) {

        console.log('Users Before Enter');

        console.log(to, from);

        next();

      }

    },

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

  ],

  linkActiveClass: 'active',

  scrollBehavior(\_, \_2, savedPosition) {

    if (savedPosition) {

      return savedPosition;

    }

    return { left:0,

             top: 0 }; // move to the top of page

  }

});

1. We can add beforeRouteUpdate to watch teamId (item in watch). BeforeRouteEnter to move different route, if BeforeRouteUpdate to move in the same parent for example from parent move to its children.

export default {

  inject: ['users', 'teams'],

  props: ['teamId'],

  components: {

    UserItem,

  },

  data() {

    return {

      teamName: '',

      members: []

   };

  },

  methods: {

    loadTeamMembers(teamId) {

    }

  },

  created() {

    // this.$route.path // /teams/t1

    this.loadTeamMembers(this.teamId);

    console.log(this.$route.query);

  },

  beforeRouteUpdate(to, from, next) {

    console.log('TeamMembers Cmp beforeRouteUpdate');

    console.log(to, from);

    next();

  },

  watch: {

    teamId(newRoute) {

      this.loadTeamMembers(newRoute);

    },

  }

};

Chapter 181 – The Global “After Each” Guard

1. In After Each we can not deny a navigation, but to send analytics data to log every navigation action and log when a user changes pages.

router.afterEach(function(to, from) {

  // Sending Analytics Data

  console.log('Global After Each');

  console.log(to, from);

});

Chapter 182 – Beyond Entering: Route Leave Guards

1. If we want to run some code on the component that is being left right before it’s being left. And you want to possibly to deny “the leave this page actioned here.” Using unmounted.

export default {

  components: {

    UserItem,

  },

  inject: ['users'],

  data() {

    return { changesSaved: false };

  },

  methods: {

    confirmInput() {

      this.$router.push('/teams');

    },

    saveChanges() {

      this.changesSaved = true;

    }

  },

  beforeRouterEnter(to, from, next) {

    next();

  },

  unmounted() {

    console.log('unmounted');

  }

};

1. If you have filled some input text on one page and you just click another navigation and move to another link, so your inputted text will be deleted. Therefore, we need one parameter saveChanges() to prevent this issue.

export default {

  components: {

    UserItem,

  },

  inject: ['users'],

  data() {

    return { changesSaved: false };

  },

  methods: {

    confirmInput() {

      this.$router.push('/teams');

    },

    saveChanges() {

      this.changesSaved = true;

    }

  },

  beforeRouterEnter(to, from, next) {

    console.log('UsersList Cmp beforeRouterEnter');

    console.log(to, from);

    next();

  },

  unmounted() {

    console.log('unmounted');

  }

};

That is only for status, then we need another component beforeRouteLeave() guard. This guard will be called before reach and before entering guards. Use confirm() to make a new message dialog to confirm you want to leave from the page or not.

export default {

  components: {

    UserItem,

  },

  inject: ['users'],

  data() {

    return { changesSaved: false };

  },

  methods: {

    confirmInput() {

      this.$router.push('/teams');

    },

    saveChanges() {

      this.changesSaved = true;

    }

  },

  beforeRouterEnter(to, from, next) {

    console.log('UsersList Cmp beforeRouterEnter');

    console.log(to, from);

    next();

  },

  beforeRouteLeave(to, from, next) {

    console.log('UsersList Cmp BeforeRouteLeave');

    console.log(to, from);

    if (this.changesSaved) {

      next();

    } else {

      const userWantsToLeave = confirm('Are you sure? You got unsaved changes!');

      next(userWantsToLeave);

    }

  },

  unmounted() {

    console.log('unmounted');

  }

};

Chapter 183 – Utilizing Route Metadata

1. We can add meta property in routes to indicate this route needs authentication, to then go to the Global BeforeEach route

const router = createRouter({

  history: createWebHistory(),

  routes: [

    { path: '/', redirect: '/teams' },

    {

      name: 'teams',

      path: '/teams',

      meta: { needsAuth: true },

      components: { default: TeamsList, footer: TeamsFooter },

      children: [

        {

          name: 'team-members',

          path: ':teamId',

          component: TeamMembers,

          props: true },

      ] }, // our-domain.com/teams => TeamsList

    {

      path: '/users',

      components: {

        default: UsersList,

        footer: UsersFooter

      },

      beforeEnter(to, from, next) {

        console.log('Users Before Enter');

        console.log(to, from);

        next();

      }

    },

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

  ],

  linkActiveClass: 'active',

  scrollBehavior(\_, \_2, savedPosition) {

    if (savedPosition) {

      return savedPosition;

    }

    return { left:0,

             top: 0 }; // move to the top of page

  }

});

And we will add some authorization code in router.beforeEach function like this.

router.beforeEach(function(to, from, next) {

  console.log('Global beforeEach');

  console.log(to, from);

  if(to.meta.needsAuth) {

    console.log('Needs auth !');

    next();

  } else {

    next();

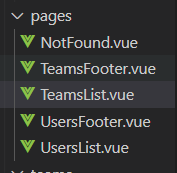
  }

  // next();

});

Chapter 184 – Organizing Route Files

1. We want to build a bigger project, we need to organize our different files and folders. We can add 1 folder named pages. Move all the template pages into folder pages.



1. Make a new router.js and fill it with routing data. Add export default router; in the end of the router.js

import UsersList from './pages/users/UsersList.vue';

import TeamMembers from './components/teams/TeamMembers.vue';

import NotFound from './pages/nav/NotFound.vue';

import TeamsFooter from './pages/teams/TeamsFooter.vue';

import UsersFooter from './pages/users/UsersFooter.vue';

const router = createRouter({

  history: createWebHistory(),

  routes: [

    { path: '/', redirect: '/teams' },

    {

      name: 'teams',

      path: '/teams',

      meta: { needsAuth: true },

      components: { default: TeamsList, footer: TeamsFooter },

      children: [

        {

          name: 'team-members',

          path: ':teamId',

          // component: TeamMembers,

          component: TeamMembers,

          props: true },

      ] }, // our-domain.com/teams => TeamsList

    {

      path: '/users',

      components: {

        default: UsersList,

        footer: UsersFooter

      },

      beforeEnter(to, from, next) {

        console.log('Users Before Enter');

        console.log(to, from);

        next();

      }

    },

    // { path: '/teams/:teamId', component: TeamMembers },

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

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

  ],

  linkActiveClass: 'active',

  // scrollBehavior(to, from, savedPosition) {}

  scrollBehavior(\_, \_2, savedPosition) {

    // console.log(to, from, savedPosition);

    if (savedPosition) {

      return savedPosition;

    }

    return { left:0,

             top: 0 }; // move to the top of page

  }

});

router.beforeEach(function(to, from, next) {

  console.log('Global beforeEach');

  console.log(to, from);

  // next(); // Agree

  // next(false); // Disagree

  // if (to.name === 'team-members') {

  //   next();

  // } else {

  //   next({name: 'team-members', params: { teamId: 't2' }});

  // }

  if(to.meta.needsAuth) {

    console.log('Needs auth !');

    next();

  } else {

    next();

  }

  // next();

});

router.afterEach(function(to, from) {

  // Sending Analytics Data

  console.log('Global After Each');

  console.log(to, from);

});

export default router;

1. Add import in main.js

import router from './router.js';

Chapter 185 – Summary

1. You can set up redirects, patches, names, metadata, multiple components that should be loaded, nested routes with children.
2. Learn how to work with dynamic path segments, to use route parameters which we can extract in the loaded components, how we can pass those route parameters as props into to be loaded components.
3. Those navigation guard and controlling the scroll behavior.
4. Learn how to navigate with router link or programmatically with the $router property and the push method.
5. Describe the location we want to to go to, for example, by setting up a string path to which we want to go, or by passing such an object using the name, adding params and possibly adding a query parameter.
6. Provide a great user experience, make sure that users don’t lose unsaved changes, you can make sure that you have URL’s and paths in the IR:’s that relate to the content that’s visible on the screen, and got sharable URL’s which you can copy and paste to load exactly that place off your Vue application, even though it’s still a single page application.