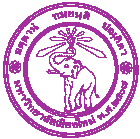
****

**CHIANG MAI UNIVERSITY**

**Bachelor of Science (Software Engineering)**

**College of Arts, Media and Technology**

**1st Semester / Academic Year 2024**

**SE 331 Component-Based Software Development**

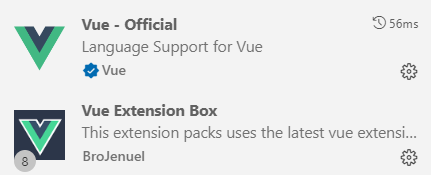
**Vue Development with Vite and TypeScript**

**Objective** In this session, The Vite, which is the tool to develop Vue application, is proposed. You will try to create a new application using the Vite setting along with the Typescript.

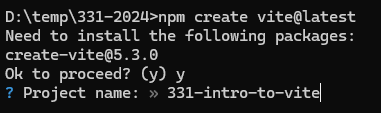
**Suggestion:** you should read the instructions step by step. Please try to answer question by question without skipping some questions that you think are extremely difficult. The step with the underline is the step in which you need to present your work to your faculty or TA. The mark is collected here. It is highly recommended that you commit to the state so you can come back to show this step again when you submit, and then you can move through the next step.

**Hint:** The +, and underline code means adding the code, and the - and the strikeout means removing the code

1. To prepare your Visual code studio, please install Vue-Official and Vue Extension Box for the Language support for Vue.

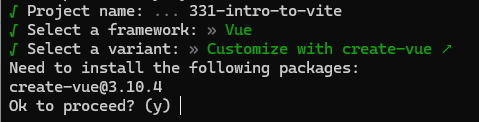


1. Create a new folder. In the folder of your project, open any command line menu (or terminal)..
   1. To confirm that you have already installed the correct Node.js version. Type node -v to see the node version, if the version is not shown or the version is less than 20, please reinstall the Node.js
   2. To run Vite, run this command. npm create vite@latest
   3. Then the prompt will ask for the project name, you can specify any name you want.

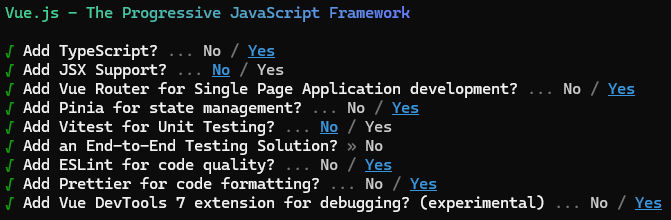


You may use any name you want

* 1. Then you select the following options

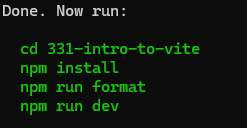


After you have proceeded, the following options should be selected



Now the scaffolding files are provided.

To run the application, type the command in your terminal



Open the web site as shown in the output screen

* 1. Show the output to the staff
  2. Create your repository, push your new created project to the repository.

**Note** that npm run format is used for formatting the source code to conform to the ease of reading.

1. Now open the vs code, update the first page to remove unwanted home component
   1. Update the HelloWorld.vue as given

<script setup lang="ts">

-defineProps<{

- msg: string

-}>()

+// defineProps<{

+// msg: string

+// }>()

</script>

<template>

- <div class="greetings">

- <h1 class="green">{{ msg }}</h1>

- <h3>

- Youâ€™ve successfully created a project with

- <a href="https://vitejs.dev/" target="\_blank" rel="noopener">Vite</a> +

- <a href="https://vuejs.org/" target="\_blank" rel="noopener">Vue 3</a>. What's next?

- </h3>

- </div>

+ <div class="greetings"></div>

</template>

-<style scoped>

-h1 {

- font-weight: 500;

- font-size: 2.6rem;

- position: relative;

- top: -10px;

-}

-

-h3 {

- font-size: 1.2rem;

-}

-

-.greetings h1,

-.greetings h3 {

- text-align: center;

-}

-

-@media (min-width: 1024px) {

- .greetings h1,

- .greetings h3 {

- text-align: left;

- }

-}

-</style>

* 1. See the output in the browser and show the staff what has been changed

1. Linking to the new Component
   1. Rename the HelloWorld.vue to EventCard.vue
   2. Remove the old HelloWorld component from App.vue

<script setup lang="ts">

import { RouterLink, RouterView } from 'vue-router'

-import HelloWorld from './components/HelloWorld.vue'

</script>

…

<div class="wrapper">

- <HelloWorld msg="You did it!" />

-

<nav>

* 1. Update main.ts to use the application css instead of the preset css

-import './assets/main.css'

+// import './assets/main.css'

* 1. Update the App.vue by removing the Vue Logo, and use our App wide CSS

<template>

- <header>

- <img alt="Vue logo" class="logo" src="@/assets/logo.svg" width="125" height="125" />

-

- <div class="wrapper">

- <nav>

- <RouterLink to="/">Home</RouterLink>

- <RouterLink to="/about">About</RouterLink>

- </nav>

- </div>

- </header>

-

- <RouterView />

+ <div id="layout">

+ <header>

+ <div class="wrapper">

+ <nav>

+ <RouterLink to="/">Home</RouterLink> |

+ <RouterLink to="/about">About</RouterLink>

+ </nav>

+ </div>

+ </header>

+

+ <RouterView />

+ </div>

</template>

<style scoped>

-header {

- line-height: 1.5;

- max-height: 100vh;

-}

-

-.logo {

- display: block;

- margin: 0 auto 2rem;

-}

-

-nav {

- width: 100%;

- font-size: 12px;

+#layout {

+ font-family: Avenir, Helvetica, Arial, sans-serif;

+ -webkit-font-smoothing: antialiased;

+ -moz-osx-font-smoothing: grayscale;

text-align: center;

- margin-top: 2rem;

+ color: #2c3e50;

}

-

-nav a.router-link-exact-active {

- color: var(--color-text);

-}

-

-nav a.router-link-exact-active:hover {

- background-color: transparent;

+nav {

+ padding: 30px;

}

-

nav a {

- display: inline-block;

- padding: 0 1rem;

- border-left: 1px solid var(--color-border);

+ font-weight: bold;

+ color: #2c3e50;

}

-

-nav a:first-of-type {

- border: 0;

-}

-

-@media (min-width: 1024px) {

- header {

- display: flex;

- place-items: center;

- padding-right: calc(var(--section-gap) / 2);

- }

-

- .logo {

- margin: 0 2rem 0 0;

- }

-

- header .wrapper {

- display: flex;

- place-items: flex-start;

- flex-wrap: wrap;

- }

-

- nav {

- text-align: left;

- margin-left: -1rem;

- font-size: 1rem;

-

- padding: 1rem 0;

- margin-top: 1rem;

- }

+nav a.router-link-exact-active {

+ color: #42b983;

}

</style>

* 1. See that the home page has been changed, now remove the unused component by remove these files

A screenshot of a computer

Description automatically generated

And then update the HomeView.vue as given

-<script setup lang="ts">

-import TheWelcome from '../components/TheWelcome.vue'

-</script>

+<script setup lang="ts"></script>

<template>

- <main>

- <TheWelcome />

- </main>

+ <div class="home"></div>

</template>

* 1. Show your output which should be like this

A screenshot of a computer

Description automatically generated

1. Now return to the EventCard.vue, setting the css for the Event card as given

<template>

- <div class="greetings"></div>

+ <div class="event-class"></div>

</template>

-<style scoped></style>

+<style scoped>

+.event-card {

+ padding: 20px;

+ width: 250px;

+ cursor: pointer;

+ border: 1px solid #39495c;

+ margin-bottom: 18px;

+}

+.event-card:hover {

+ transform: scale(1.01);

+ box-shadow: 0 3px 12px 0 rgba(0, 0, 0, 0.2);

+}

+</style>

1. Now update the EventCard.vue to show an event

<script setup lang="ts">

+import { ref } from 'vue'

// defineProps<{

// msg: string

// }>()

+const event = ref({

+ id: 5928101,

+ category: 'animal welfare',

+ title: 'Cat Adoption Day',

+ description: 'Find your new feline friend at this event.',

+ location: 'Meow Town',

+ date: 'January 28, 2022',

+ time: '12:00',

+ petsAllowed: true,

+ organizer: 'Kat Laydee'

+})

</script>

<template>

- <div class="event-class"></div>

+ <div class="event-class">

+ <div class="event-card">

+ <h2>{{ event.title }}</h2>

+ <span>@{{ event.time }} on {{ event.date }}</span>

+ </div>

+ </div>

</template>

1. To see the event card, update the HomeView.vue to use the EventCard component as given

-<script setup lang="ts"></script>

+<script setup lang="ts">

+import EventCard from '@/components/EventCard.vue'

+</script>

<template>

- <div class="home"></div>

+ <div class="home">

+ <EventCard />

+ </div>

</template>

Now show the output that you have done

1. Now we want our HomeView to show more than one Event at a time,
   1. Update the EventCard.vue to receive the event object via Props instead of defining the object in the event card

<script setup lang="ts">

import { ref } from 'vue'

-

-// defineProps<{

-// msg: string

-// }>()

-const event = ref({

- id: 5928101,

- category: 'animal welfare',

- title: 'Cat Adoption Day',

- description: 'Find your new feline friend at this event.',

- location: 'Meow Town',

- date: 'January 28, 2022',

- time: '12:00',

- petsAllowed: true,

- organizer: 'Kat Laydee'

-})

+ import type { Event } from '@/types'

+defineProps<{

+ event: Event

+}>()

+// const event = ref({

+// id: 5928101,

+// category: 'animal welfare',

+// title: 'Cat Adoption Day',

+// description: 'Find your new feline friend at this event.',

+// location: 'Meow Town',

+// date: 'January 28, 2022',

+// time: '12:00',

+// petsAllowed: true,

+// organizer: 'Kat Laydee'

+// })

</script>

* 1. Notice that we have define the type Event, to declare the type create the file types.ts in the src folder

A screenshot of a computer

Description automatically generated

* 1. The file types.ts provides the type declaration. The type of event is provided as given

+export interface Event {

+ id: number

+ category: string

+ title: string

+ description: string

+ location: string

+ date: string

+ time: string

+ petsAllowed: boolean

+ organizer: string

+}

* 1. Then update the HomeView.vue to provide the list of Event objects and inject them to the EventCard component using for loop.

First provide the event list

<script setup lang="ts">

import EventCard from '@/components/EventCard.vue'

+import type { Event } from '@/types'

+import { ref } from 'vue'

+const events = ref<Event[]>([

+ {

+ id: 5928101,

+ category: 'animal welfare',

+ title: 'Cat Adoption Day',

+ description: 'Find your new feline friend at this event.',

+ location: 'Meow Town',

+ date: 'January 28, 2022',

+ time: '12:00',

+ petsAllowed: true,

+ organizer: 'Kat Laydee'

+ },

+ {

+ id: 4582797,

+ category: 'food',

+ title: 'Community Gardening',

+ description: 'Join us as we tend to the community edible plants.',

+ location: 'Flora City',

+ date: 'March 14, 2022',

+ time: '10:00',

+ petsAllowed: true,

+ organizer: 'Fern Pollin'

+ },

+ {

+ id: 8419988,

+ category: 'sustainability',

+ title: 'Beach Cleanup',

+ description: 'Help pick up trash along the shore.',

+ location: 'Playa Del Carmen',

+ date: 'July 22, 2022',

+ time: '11:00',

+ petsAllowed: false,

+ organizer: 'Carey Wales'

+ }

+])

</script>

Note that the list of the event object is provided for you to copy and paste in the file events.json in the e-learning system.

Then update the template to use v-for for looping the Event list

<template>

<div class="home">

- <EventCard />

+ <EventCard v-for="event in events" :key="event.id" :event="event" />

</div>

</template>

Now show the result of web browser to the staff

* 1. Update the card to show at the center of the page, update the HomeView.vue by adding the css as given

<template>

- <div class="home">

+ <div class="events">

<EventCard v-for="event in events" :key="event.id" :event="event" />

</div>

</template>

+

+<style scoped>

+.events {

+ display: flex;

+ flex-direction: column;

+ align-items: center;

+}

+</style>

* 1. Then update the css in the App.js to be the Global style by removing the scoped in the style of the App.vue

</div>

</template>

-<style scoped>

+<style>

#layout {

nav a.router-link-exact-active {

color: #42b983;

}

+

+h2 {

+ font-size: 20px;

+}

</style>

Now the layout should show properly like this

A screenshot of a phone

Description automatically generated

* 1. Create a new component that shows only the categories and organizer name. Both of them should be the same font size, and both text should align right. Add the new component into the Home view and show the new components using the events in the data.

1. Now we want to have the title for the HomeView
   1. Add the text to show in the App.vue

</div>

</header>

-

+ <h1>Events For Good</h1>

+ <!-- new element -->

<RouterView />

</div>

</template>

The problem is when we click the About Menu, the word “Events for Good” is still shown which is not the required behavior

* 1. So we move the heading text from App.vue to

</nav>

</div>

</header>

- <h1>Events For Good</h1>

- <!-- new element -->

<RouterView />

</div>

* 1. Then move to the HomeView.vue

<template>

+ <h1>Events For Good</h1>

+ <!-- new element -->

<div class="events">

<EventCard v-for="event in events" :key="event.id" :event="event" />

</div>

Now when we move to about page, we will not see the Header again.

Show this to the staff

1. Remove the css style in AboutView.vue for make it simple for our application

<style>

-@media (min-width: 1024px) {

+/\* @media (min-width: 1024px) {

.about {

min-height: 100vh;

display: flex;

align-items: center;

}

-}

+} \*/

</style>

1. To make the component easy to read, rename the HomeView.vue to EventListView.vue
   1. Update the src/router/index.ts to link to the EventListView.vue instead of the old HomeView.vue

import { createRouter, createWebHistory } from 'vue-router'

-import HomeView from '../views/HomeView.vue'

+import EventListView from '../views/EventListView.vue'

const router = createRouter({

history: createWebHistory(import.meta.env.BASE\_URL),

routes: [

{

path: '/',

- name: 'home',

- component: HomeView

+ name: 'event-list-view',

+ component: EventListView

},

{

path: '/about',

1. We reconfigure the AboutView.vue to match without application
   1. Update the content of the AboutView.vue

<template>

<div class="about">

- <h1>This is an about page</h1>

+ <h1>A site for events to better the world.</h1>

</div>

</template>

* 1. Check on the src/router/index.ts update technique to load the about page as given

import { createRouter, createWebHistory } from 'vue-router'

-import EventListView from '../views/EventListView.vue'

-

+import EventListView from '@/views/EventListView.vue'

+import AboutView from '@/views/AboutView.vue'

const router = createRouter({

history: createWebHistory(import.meta.env.BASE\_URL),

routes: [

…

{

path: '/about',

name: 'about',

- // route level code-splitting

- // this generates a separate chunk (About.[hash].js) for this route

- // which is lazy-loaded when the route is visited.

- component: () => import('../views/AboutView.vue')

+ component: AboutView

}

Why we do this, I will be explained in the lecture class.

1. Now, we will change the menu from Home to Event. Update App.vue with the given information

<div class="wrapper">

<nav>

- <RouterLink to="/">Home</RouterLink> |

+ <RouterLink to="/">Event</RouterLink> |

<RouterLink to="/about">About</RouterLink>

</nav>

</div>

1. Provide the mock server
   1. The real application should load data from the backend. As of now, we have not implemented any web server yet. The mock server can be provided by <https://my-json-server.typicode.com/>
   2. Download db.json file from the e-learning system.
   3. Create the new repository using your id account, then copy the db.json to your repository and push the code to the Github server

**Note** that your repository must be the public repository

* 1. Create your mock server by calling https://my-json-server.typicode.com/[your username]/[your repo name]/events in the browser,

You should see something similar to this

A screenshot of a computer code

Description automatically generated

* 1. Add a new event object in your database then show the staff in the browser that you have add a new event object in the database

1. Now we will prepare to load the data from the server
   1. Install the Axios to the project for loading data from the server using the command.   
       npm install axios
   2. Remove the old data from the EventListview.vue

import EventCard from '@/components/EventCard.vue'

import type { Event } from '@/types'

import { ref } from 'vue'

-const events = ref<Event[]>([

- {

- id: 5928101,

- category: 'animal welfare',

- title: 'Cat Adoption Day',

- description: 'Find your new feline friend at this event.',

- location: 'Meow Town',

- date: 'January 28, 2022',

- time: '12:00',

- petsAllowed: true,

- organizer: 'Kat Laydee'

- },

- {

- id: 4582797,

- category: 'food',

- title: 'Community Gardening',

- description: 'Join us as we tend to the community edible plants.',

- location: 'Flora City',

- date: 'March 14, 2022',

- time: '10:00',

- petsAllowed: true,

- organizer: 'Fern Pollin'

- },

- {

- id: 8419988,

- category: 'sustainability',

- title: 'Beach Cleanup',

- description: 'Help pick up trash along the shore.',

- location: 'Playa Del Carmen',

- date: 'July 22, 2022',

- time: '11:00',

- petsAllowed: false,

- organizer: 'Carey Wales'

- }

-])

+const events = ref<Event[]>(null)

</script>

* 1. Then add the axios to load data from mock server we have created, update the EventListView.vue as given

<script setup lang="ts">

import EventCard from '@/components/EventCard.vue'

import type { Event } from '@/types'

-import { ref } from 'vue'

+import { ref, onMounted } from 'vue'

+import axios from 'axios'

+

const events = ref<Event[]>(null)

+

+onMounted(() => {

+ axios

+ .get('[your mock server url]')

+ .then((response) => {

+ console.log(response.data)

+ })

+ .catch((error) => {

+ console.error('There was an error!', error)

+ })

+})

</script>

When it run show the output in the console in development tools to the staff

* 1. Now we will use the received data to show in the user interface, update the EventListView.vue as given

onMounted(() => {

axios

.get('[your mock server url]')

.then((response) => {

- console.log(response.data)

+ events.value = response.data

})

Then see the output

1. To make the code clear, we should refactor the code to separate the services part from the view part.  
   1. Create a new folder named services, and create the file EventService.ts in the services folder. Add the given content to the file.

+import axios from 'axios'

+

+const apiClient = axios.create({

+ baseURL: '[your mock server url]',

+ withCredentials: false,

+ headers: {

+ Accept: 'application/json',

+ 'Content-Type': 'application/json'

+ }

+})

+

+export default {

+ getEvents() {

+ return apiClient.get('/events')

+ }

+}

* 1. Modified the EventListView.vue by importing the EventService instead of axios

import { ref, onMounted } from 'vue'

-import axios from 'axios'

+import EventService from '@/services/EventService'

const events = ref<Event[]>(null)

onMounted(() => {

- axios

- .get('[your mock server url]')

+ EventService.getEvents()

.then((response) => {

events.value = response.data

})

Now see the output, it should display as the same but our code looks a lot cleaner.

* 1. Create a new service to get the students information given here (<https://dv-student-backend-2019.appspot.com/students>). Then create a new view to show the student name, surname, and gpa on the cards..