MINAB
IT SOLUTIONS

Frontend development Class 04, Series 02

Frontend

Hahu Jobs

CLASS 04

Hahu Jobs

01 Single file components

02 Components in-depth

03 Reusability

04 Routing

05 Hands on vue

Component registration

Global Registration

We can make components available globally in the current Vue application using the app.component() method:

```
import MyComponent from './App.vue'

app.component('MyComponent', MyComponent)
```

Local Registration

Local registration scopes the availability of the registered components to the current component only. It makes the dependency relationship more explicit, and is more tree-shaking friendly.

When using SFC with <script se ally registered locally:

```
<script setup>
import ComponentA from './ComponentA.vue'
</script>

<template>
  <ComponentA />
</template>
```

MINAB IT SOLUTIONS

HahuJobs

Props

Prop Validation

Components can specify requirements for their props, such as the types you've already seen. If a requirement is not met, Vue will warn you in the browser's JavaScript console.

```
defineProps({
  // Basic type check
  //  (`null` and `undefined` values will allow any type)
  propA: Number,
  // Multiple possible types
  propB: [String, Number],
  // Required string
  propC: {
    type: String,
    required: true
  },
  // Number with a default value
  propD: {
    type: Number,
    default: 100
  },
  // Object with a default value
  propE: {
    type: Object,
    // Object or array defaults must be returned from
    // a factory function
    default() {
      return { message: 'hello' }
    }
  },
  // Custom validator function
```

Props

Prop Validation

Components can specify requirements for their props, such as the types you've already seen. If a requirement is not met, Vue will warn you in the browser's JavaScript console.

```
defineProps({
  // Basic type check
  //  (`null` and `undefined` values will allow any type)
  propA: Number,
  // Multiple possible types
  propB: [String, Number],
  // Required string
  propC: {
    type: String,
    required: true
  },
  // Number with a default value
  propD: {
    type: Number,
    default: 100
  },
  // Object with a default value
  propE: {
    type: Object,
    // Object or array defaults must be returned from
    // a factory function
    default() {
      return { message: 'hello' }
    }
  },
  // Custom validator function
```

Events

Event Arguments

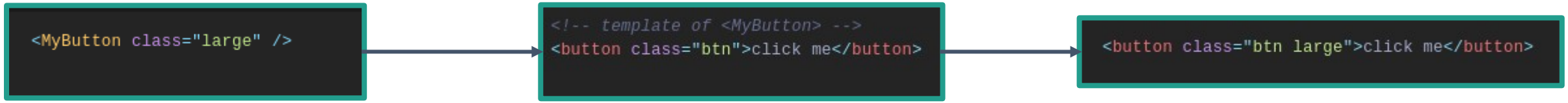
Components can specify requirements for their props, such as the types you've already seen. If a requirement is not met, Vue will warn you in the browser's JavaScript console.

```
defineProps({
  // Basic type check
  //  (`null` and `undefined` values will allow any type)
  propA: Number,
  // Multiple possible types
  propB: [String, Number],
  // Required string
  propC: {
    type: String,
    required: true
  },
  // Number with a default value
  propD: {
    type: Number,
    default: 100
  },
  // Object with a default value
  propE: {
    type: Object,
    // Object or array defaults must be returned from
    // a factory function
    default() {
      return { message: 'hello' }
    }
  },
  // Custom validator function
```

Fallthrough attributes

Fallthrough Attributes

A "fallthrough attribute" is an attribute or v-on event listener that is passed to a component, but is not explicitly declared in the receiving component's props or emits. Common examples of this include class, style, and id attributes.



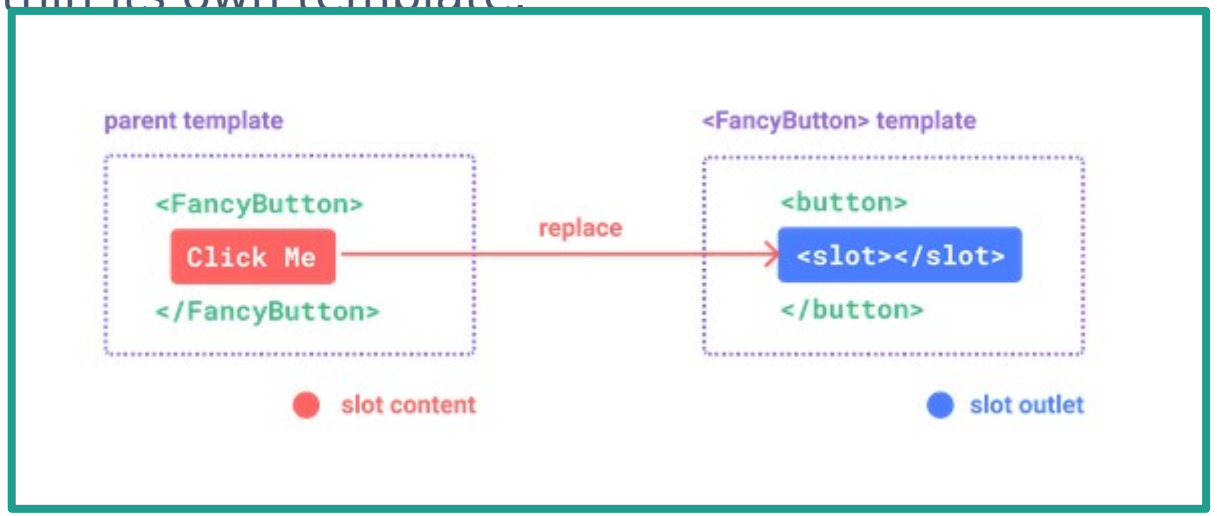
Slots

Slot Content and Outlet

We have learned that components can accept props, which can be JavaScript values of any type. But how about template content? In some cases, we may want to pass a template fragment to a child component, and let the child component render the fragment within its own template.

```
<FancyButton>
  Click me! <!-- slot content -->
</FancyButton>
```

```
<button class="fancy-btn">
  <slot></slot> <!-- slot outlet -->
</button>
```



MINAB IT SOLUTIONS

HahuJobs

Slots

Render Scope

Slot content has access to the data scope of the parent component, because it is defined in the parent. For example:

```
<span>{{ message }}</span>
<FancyButton>{{ message }}</FancyButton>
```

Slot content does **not** have access to the child component's data.
As a rule, remember that:

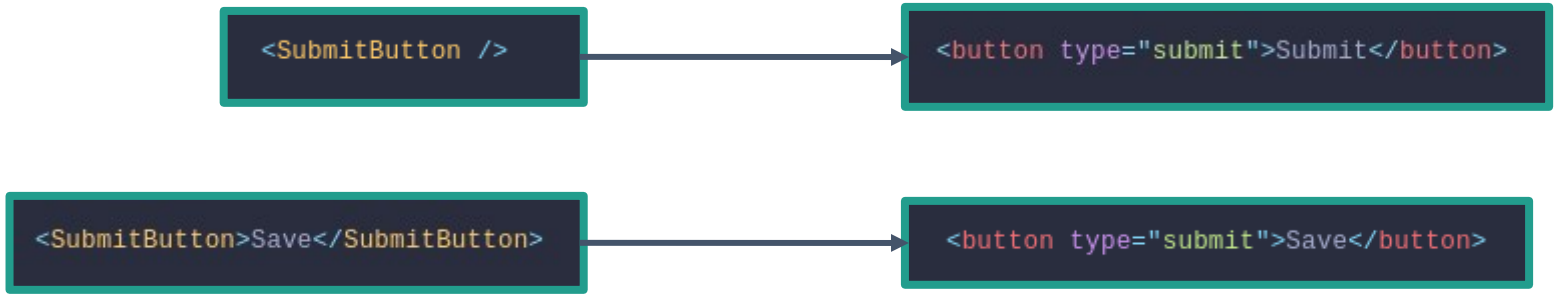
Everything in the parent template is compiled in parent scope; everything in the child template is compiled in the child scope.

Slots

Fallback Content

There are cases when it's useful to specify fallback (i.e. default) content for a slot, to be rendered only when no content is provided.

```
<button type="submit">
  <slot>
    Submit <!-- fallback content -->
  </slot>
</button>
```



Slots

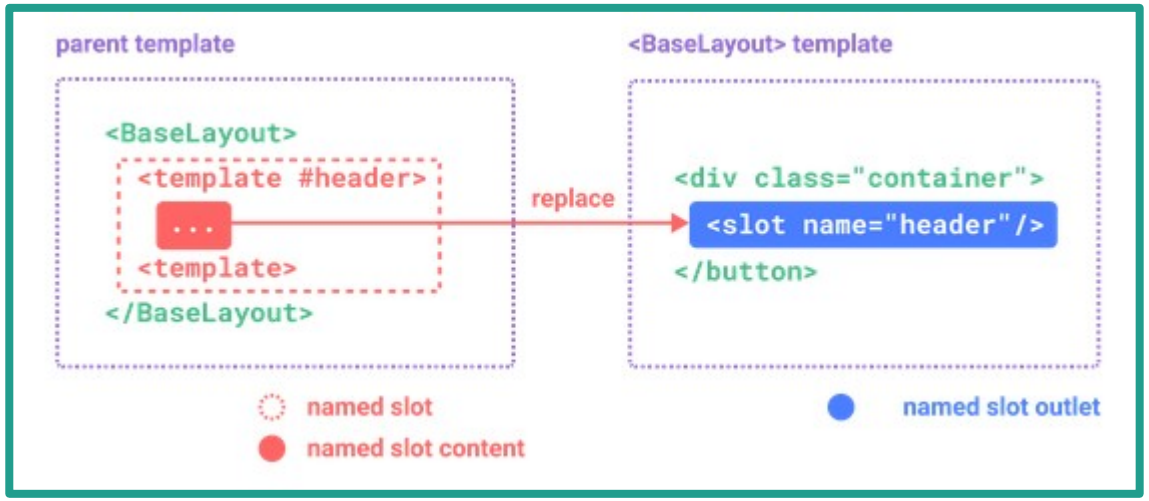
Named Slots

There are times when it's useful to have multiple slot outlets in a single component.

```
<BaseLayout>
  <template #header>
    <h1>Here might be a page title</h1>
  </template>

  <template #default>
    <p>A paragraph for the main content.</p>
    <p>And another one.</p>
  </template>

  <template #footer>
    <p>Here's some contact info</p>
  </template>
</BaseLayout>
```




```
<BaseLayout>
  <template v-slot:header>
    <!-- content for the header slot -->
  </template>
</BaseLayout>
```

Slots

Scoped Slots

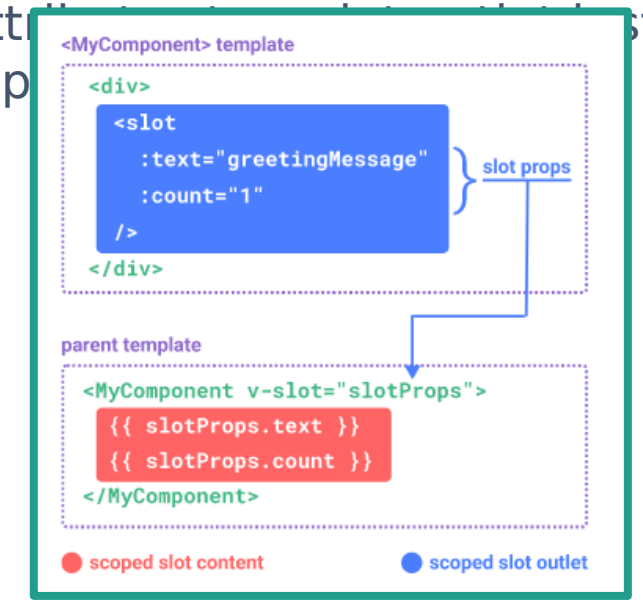
As discussed in Render Scope, slot content does not have access to state in the child component.

However, there are cases where it could be useful if a slot's content can make use of data from both the parent scope and the child scope. To achieve that, we need a way for the child to pass data to a slot when rendering it.

In fa pass att st like passing a comp

```
<!-- <MyComponent> template -->
<div>
  <slot :text="greetingMessage" :count="1"></slot>
</div>
```

```
<MyComponent v-slot="slotProps">
  {{ slotProps.text }} {{ slotProps.count }}
</MyComponent>
```



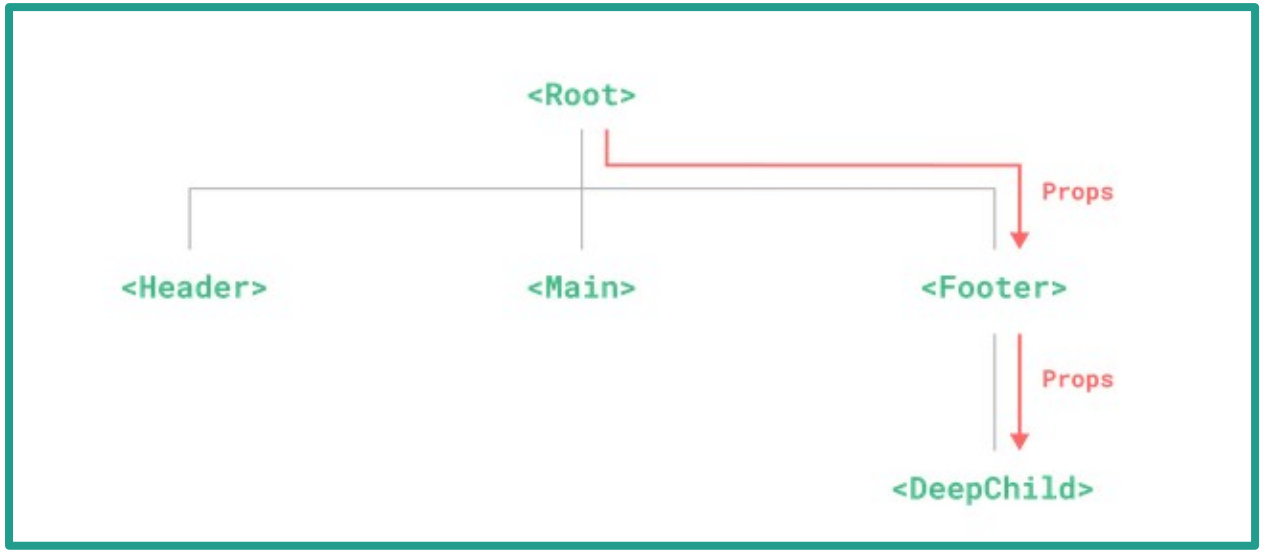
MINAB IT SOLUTIONS

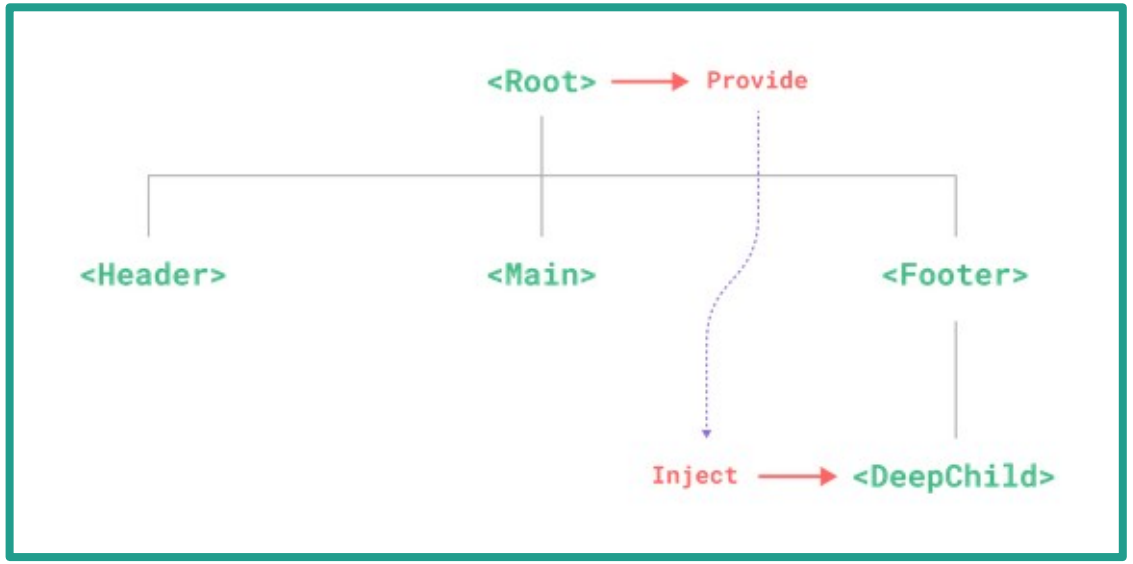
HahuJobs

Provide / Inject

Prop Drilling

Usually, when we need to pass data from the parent to a child component, we use props. However, imagine the case where we have a large component tree, and a deeply nested component needs something from a distant ancestor component.





MINAB IT SOLUTIONS

HahuJobs

Provide / Inject

Provide

To provide data to a component's descendants, use the provide() function:

```
<script setup>
import { provide } from 'vue'

provide(/* key */ 'message', /* value */ 'hello!')
</script>
```

App-level Provide

In addition to providing data in a component, we can also provide at the app level:

```
import { createApp } from 'vue'

const app = createApp({})

app.provide(/* key */ 'message', /* value */ 'hello!')
```

Provide / Inject

Inject

To inject data provided by an ancestor component, use the inject() function:

```
<script setup>
import { inject } from 'vue'

const message = inject('message')
</script>
```

Injection Default Values

By default, inject assumes that the injected key is provided somewhere in the parent chain.

```
// `value` will be "default value"
// if no data matching "message" was provided
const value = inject('message', 'default value')
```

Reusability

Composables

In the context of Vue applications, a "composable" is a function that leverages Vue Composition API to encapsulate and reuse stateful logic.

When building frontend applications, we often have the need to reuse logic for common tasks.

Mouse Tracker Example

If we were to implement the mouse tracking functionality using Composition API directly inside a component, it would look like this:

```
<script setup>
import { ref, onMounted, onUnmounted } from 'vue'

const x = ref(0)
const y = ref(0)

function update(event) {
  x.value = event.pageX
  y.value = event.pageY
}

onMounted(() => window.addEventListener('mousemove', update))
onUnmounted(() => window.removeEventListener('mousemove', update))
</script>

<template>Mouse position is at: {{ x }}, {{ y }}</template>
```

Reusability

Mouse Tracker Example

But what if we want to reuse the same logic in multiple components? We can extract the logic into an external file, as a composable function:

```
// mouse.js
import { ref, onMounted, onUnmounted } from 'vue'

// by convention, composable function names start with "use"
export function useMouse() {
  // state encapsulated and managed by the composable
  const x = ref(0)
  const y = ref(0)

  // a composable can update its managed state over time.
  function update(event) {
    x.value = event.pageX
    y.value = event.pageY
  }

  // a composable can also hook into its owner component's
  // lifecycle to setup and teardown side effects.
  onMounted(() => window.addEventListener('mousemove', update))
  onUnmounted(() => window.removeEventListener('mousemove', update))

  // expose managed state as return value
  return { x, y }
}
```

And this is how it can be used in components:

```
<script setup>
import { useMouse } from './mouse.js'

const { x, y } = useMouse()
</script>

<template>Mouse position is at: {{ x }}, {{ y }}</template>
```

Reusability

Custom Directives

We have introduced two forms of code reuse in Vue: components and composables. Components are the main building blocks, while composables are focused on reusing stateful logic. Custom directives, on the other hand, are mainly intended for reusing logic that involves low-level DOM access on plain elements.

```
<script setup>
// enables v-focus in templates
const vFocus = {
  mounted: (el) => el.focus()
}
</script>

<template>
  <input v-focus />
</template>
```

In <script setup>, any camelCase variable that starts with the **v prefix** can be used as a custom directive. In the example above, vFocus can be used in the template as v-focus.

Reusability

Plugins

Plugins are self-contained code that usually add app-level functionality to Vue. This is how we install a plugin:

```
import { createApp } from 'vue'

const app = createApp({})

app.use(myPlugin, {
  /* optional options */
})
```

A plugin is defined as either an object that exposes an install() method, or simply a function that acts as the install function itself. The install function receives the app instance along with additional options passed to app.use(), if any:

```
const myPlugin = {
  install(app, options) {
    // configure the app
  }
}
```

Reusability

Plugins

Plugins are self-contained code that usually add app-level functionality to Vue. This is how we install a plugin:

1. Register one or more global components or custom directives with app.component() and app.directive().
2. Make a resource injectable throughout the app by calling app.provide().
3. Add some global instance properties or methods by attaching them to app.config.globalProperties.
4. A library that needs to perform some combination of the above (e.g. vue-router).

Routing

Vue Router

Vue Router is the official routing library for Vue.js applications. Even though you can go ahead and use other generic routing libraries, Vue Router deeply integrates with the ideology of Vue.js to make Single Page Applications (SPAs) easy to build.

- Nested routes mapping
- Dynamic Routing
- Modular, component-based router configuration
- Route params, query, wildcards
- View transition effects powered by Vue.js' transition system
- Fine-grained navigation control
- Links with automatic active CSS classes
- HTML5 history mode or hash mode
- Customizable Scroll Behavior
- Proper encoding for URLs

MINAB IT SOLUTIONS

HahuJobs

Routing

Installation

To install vue router in our application we must run the following command

```
npm install vue-router@4
```

MINAB IT SOLUTIONS

HahuJobs

Routing

Setup

Creating a Single-page Application with Vue + Vue Router feels natural: with Vue.js, we are already composing our application with components. When adding Vue Router to the mix, all we need to do is map our components to the routes and let Vue Router know where to render them.

```
const routes = [
  { path: '/', component: Home },
  { path: '/about', component: About },
]

// 3. Create the router instance and pass the `routes` option
// You can pass in additional options here, but let's
// keep it simple for now.
const router = VueRouter.createRouter({
  // 4. Provide the history implementation to use. We are using
  history: VueRouter.createWebHashHistory(),
  routes, // short for `routes: routes`
})

// 5. Create and mount the root instance.
const app = Vue.createApp({})
// Make sure to _use_ the router instance to make the
// whole app router-aware.
app.use(router)

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

```
<script src="https://unpkg.com/vue@3"></script>
<script src="https://unpkg.com/vue-router@4"></script>

<div id="app">
  <h1>Hello App!</h1>
  <p>
    <!-- use the router-link component for navigation. -->
    <!-- specify the link by passing the `to` prop. -->
    <!-- `<router-link>` will render an `<a>` tag with the correct `href`
    <router-link to="/">Go to Home</router-link>
    <router-link to="/about">Go to About</router-link>
  </p>
  <!-- route outlet -->
  <!-- component matched by the route will render here -->
  <router-view></router-view>
</div>
```

MINAB IT SOLUTIONS

HahuJobs

Routing

Setup

Creating a Single-page Application with Vue + Vue Router feels natural: with Vue.js, we are already composing our application with components. When adding Vue Router to the mix, all we need to do is map our components to the routes and let Vue Router know where to render them.

```
// 1. Define route components.
// These can be imported from other files
const Home = { template: '<div>Home</div>' }
const About = { template: '<div>About</div>' }

// 2. Define some routes
// Each route should map to a component.
// We'll talk about nested routes later.
const routes = [
  { path: '/', component: Home },
  { path: '/about', component: About },
]

// 3. Create the router instance and pass the `routes` option
// You can pass in additional options here, but let's
// keep it simple for now.
const router = VueRouter.createRouter({
  // 4. Provide the history implementation to use. We are using
  history: VueRouter.createWebHashHistory(),
  routes, // short for `routes: routes`
})

// 5. Create and mount the root instance.
const app = Vue.createApp({})
// Make sure to _use_ the router instance to make the
// whole app router-aware.
app.use(router)

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

```
<script src="https://unpkg.com/vue@3"></script>
<script src="https://unpkg.com/vue-router@4"></script>

<div id="app">
  <h1>Hello App!</h1>
  <p>
    <!-- use the router-link component for navigation. -->
    <!-- specify the link by passing the `to` prop. -->
    <!-- `<router-link>` will render an `<a>` tag with the correct `href`
    <router-link to="/">Go to Home</router-link>
    <router-link to="/about">Go to About</router-link>
  </p>
  <!-- route outlet -->
  <!-- component matched by the route will render here -->
  <router-view></router-view>
</div>
```

Routing

router-link & router-view

router-link

Note how instead of using regular a tags, we use a custom component router-link to create links. This allows Vue Router to change the URL without reloading the page, handle URL generation as well as its encoding.

router-view

router-view will display the component that corresponds to the url. You can put it anywhere to adapt it to your layout.

Routing

Programmatic Navigation

Aside from using <router-link> to create anchor tags for declarative navigation, we can do this programmatically using the router's instance methods.

Declarative	Programmatic
`<router-link :to="...">`	`router.push(...)`

```
// literal string path
router.push('/users/eduardo')

// object with path
router.push({ path: '/users/eduardo' })

// named route with params to let the router build the url
router.push({ name: 'user', params: { username: 'eduardo' } })

// with query, resulting in /register?plan=private
router.push({ path: '/register', query: { plan: 'private' } })

// with hash, resulting in /about#team
router.push({ path: '/about', hash: '#team' })
```

Routing

Vue Router

Read more at https://router.vuejs.org

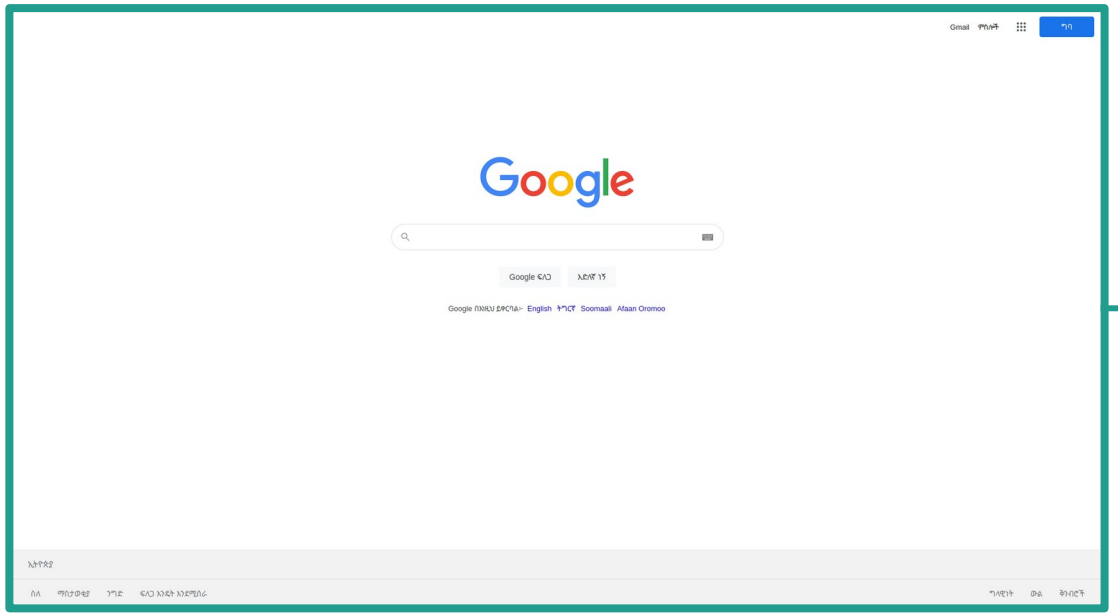
MINAB IT SOLUTIONS

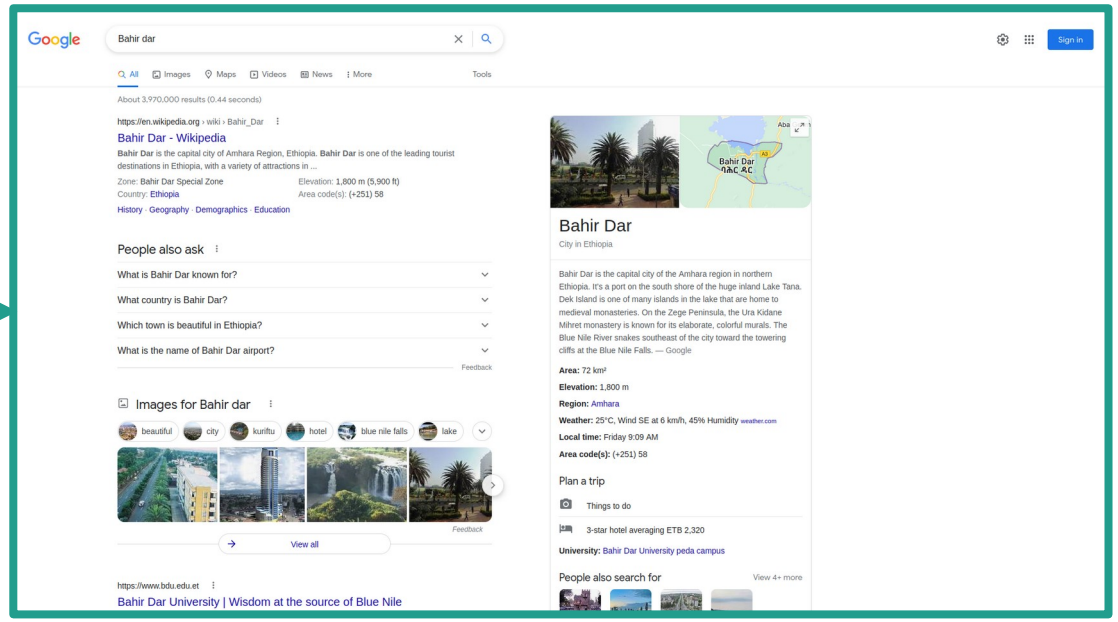
HahuJobs

Routing

Hands on Vue

Add router to your existing google pages & refactor your components with today’s lesson





MINAB IT SOLUTIONS

Hahu Jobs

Reading Assignment

- Figma
- Tailwind CSS
- Vue
- Vite
- Network requests
- Graphql
- apollo client for vue

MINAB IT SOLUTIONS

Hahu Jobs

MINAB
IT SOLUTIONS

Thank you!

Hahu Jobs

ለሀገር ልጅ በሀገር ልጅ !

Michael Sahlu
michael.sahlu@hahu.jobs