MINAB
IT SOLUTIONS

Frontend development Class 03, Series 02

Frontend

HahuJobs

CLASS 03

Hahu Jobs

01 Single page apps

02 Javascript frameworks

03 Vuejs

04 Basics of vue

05 Setting up vue

06 Hands on vue

Single page apps

A single-page application (SPA) is a web application or website that interacts with the user by dynamically rewriting the current web page with new data from the web server, instead of the default method of a web browser loading entire new pages.

The goal is faster transitions that make the website feel more like a native app.

In a SPA, a page refresh never occurs; instead, all necessary HTML, JavaScript, and CSS code is either retrieved by the browser with a single page load,[1] or the appropriate resources are dynamically loaded and added to the page as necessary, usually in response to user actions.

Javascript frameworks

JS frameworks are JavaScript code libraries that have pre-written code to use for routine programming features and tasks. It is literally a framework to build websites or web applications around.

For example, in plain JS, you have to manually update the DOM using DOM APIs for setting styles updating content, etc.

JS frameworks can help automate this repetitive task using features like 2-way binding and templating.

Javascript frameworks

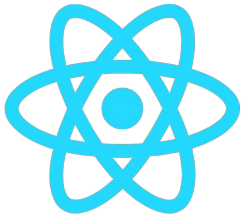
Should you use one?

Depends on what you are trying to build!

Top javascript frameworks in 2022 are

Vue.js

Star 195k



React

Star 195k

Svelte

Star 57.4k

Angular.j

Star 80.7k

MINAB IT SOLUTIONS

HahuJobs

Vue.js

Vue.js

Star 195k

An approachable, performant and versatile framework for building web user interfaces.

Vue is a JavaScript framework for building user interfaces. It builds on top of standard HTML, CSS and JavaScript, and provides a declarative and component-based programming model that helps you efficiently develop user interfaces, be it simple or complex.

MINAB IT SOLUTIONS

HahuJobs

Vue.js

Creating a Vue Application

Every Vue application starts by creating a new **application instance** with the createApp function:

```
import { createApp } from 'vue'

const app = createApp({
  /* root component options */
})
```

Mounting the app

An application instance won't render anything until its .mount() method is called.

```
<div id="app"></div>
```

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

Vue.js

Template syntax

Vue uses an HTML-based template syntax that allows you to declaratively bind the rendered DOM to the underlying component instance's data.

Text Interpolation

The most basic form of data binding is text interpolation using the "Mustache" syntax (double curly braces):

```
<span>Message: {{ msg }}</span>
```

Vue.js

Template syntax

Vue uses an HTML-based template syntax that allows you to declaratively bind the rendered DOM to the underlying component instance's data.

Raw HTML

The double mustaches interprets the data as plain text, not HTML. In order to output real HTML, you will need to use the v-html directive:

```
<p>Using text interpolation: {{ rawHtml }}</p>
<p>Using v-html directive: <span v-html="rawHtml"></span></p>
```

> ⚠ Security Warning
>
> Dynamically rendering arbitrary HTML on your website can be very dangerous because it can easily lead to **XSS vulnerabilities**. Only use `v-html` on trusted content and never on user-provided content.

MINAB IT SOLUTIONS

HahuJobs

Vue.js

Template syntax

Vue uses an HTML-based template syntax that allows you to declaratively bind the rendered DOM to the underlying component instance's data.

Attribute Bindings

Mustaches cannot be used inside HTML attributes. Instead, use a v-bind directive:

```
<div v-bind:id="dynamicId"></div>
```

Because v-bind is so commonly used, it has a dedicated **shorthand** syntax:

```
<div :id="dynamicId"></div>
```

Vue.js

API Preference

Vue components can be authored in two different API styles: **Options API** and **Composition API.**

Options API

With Options API, we define a component's logic using an object of options such as data, methods, and mounted.

```
<script>
export default {
  // Properties returned from data() becomes reactive state
  // and will be exposed on `this`.
  data() {
    return {
      count: 0
    }
  },

  // Methods are functions that mutate state and trigger updates.
  // They can be bound as event listeners in templates.
  methods: {
    increment() {
      this.count++
    }
  },

  // Lifecycle hooks are called at different stages
  // of a component's lifecycle.
  // This function will be called when the component is mounted.
  mounted() {
    console.log(`The initial count is ${this.count}.`)
  }
}
</script>

<template>
  <button @click="increment">Count is: {{ count }}</button>
</template>
```

Compositio n API

With Composition API, we define a component's logic using imported API functions. In SFCs, Composition API is typically used with <script setup>.

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

// reactive state
const count = ref(0)

// functions that mutate state and trigger updates
function increment() {
  count.value++
}

// lifecycle hooks
onMounted(() => {
  console.log(`The initial count is ${count.value}.`)
})
</script>

<template>
  <button @click="increment">Count is: {{ count }}</button>
</template>
```

MINAB IT SOLUTIONS

HahuJobs

Vue.js

Reactivity Fundamentals

Reactivity, among JavaScript frameworks, is the phenomenon in which changes in the application state are automatically reflected in the DOM

```
let A0 = 1
let A1 = 2
let A2 = A0 + A1

console.log(A2) // 3

A0 = 2
console.log(A2) // Still 3
```

Vue.js

Reactivity Fundamentals

Reactivity, among JavaScript frameworks, is the phenomenon in which changes in the application state are automatically reflected in the DOM

Reactive object

Reactive objects are JavaScript Proxies and behave just like normal objects. The difference is that Vue is able to track the property access and mutations of a reactive object.

```
import { reactive } from 'vue'

const state = reactive({ count: 0 })
```

Limitations of reactive()

1) It only works for object types (objects, arrays, and collection types such as Map and Set). It cannot hold primitive types such as string, number or boolean.
2) Since Vue's reactivity tracking works over property access, we must always keep the same reference to the reactive object. This means we can't easily "replace" a reactive object because the reactivity connection to the first reference is lost:

Vue.js

Reactivity Fundamentals

Reactivity, among JavaScript frameworks, is the phenomenon in which changes in the application state are automatically reflected in the DOM

Reactive Variables with ref()

To address the limitations of reactive(), Vue also provides a ref() function which allows us to create reactive "refs" that can hold any value type:

```
import { ref } from 'vue'

const count = ref(0)
```

ref() takes the argument and returns it wrapped within a ref object with

```
const count = ref(0)

console.log(count) // { value: 0 }
console.log(count.value) // 0

count.value++
console.log(count.value) // 1
```

MINAB IT SOLUTIONS

HahuJobs

Vue.js

Computed Properties

In-template expressions are very convenient, but they are meant for simple operations. Putting too much logic in your templates can make them bloated and hard to maintain. For example, if we have an object with a nested array:

```
const author = reactive({
  name: 'John Doe',
  books: [
    'Vue 2 - Advanced Guide',
    'Vue 3 - Basic Guide',
    'Vue 4 - The Mystery'
  ]
})
```

```
<p>Has published books:</p>
<span>{{ author.books.length > 0 ? 'Yes' : 'No' }}</span>
```

VS

```
<script setup>
import { reactive, computed } from 'vue'

const author = reactive({
  name: 'John Doe',
  books: [
    'Vue 2 - Advanced Guide',
    'Vue 3 - Basic Guide',
    'Vue 4 - The Mystery'
  ]
})

// a computed ref
const publishedBooksMessage = computed(() => {
  return author.books.length > 0 ? 'Yes' : 'No'
})
</script>

<template>
  <p>Has published books:</p>
  <span>{{ publishedBooksMessage }}</span>
</template>
```

Vue.js

Class and Style Bindings

Since they are both attributes, we can use v-bind to handle them: we only need to calculate a final string with our expressions. However, meddling with string concatenation is annoying and error-prone.

For this reason, Vue provides special enhancements when **v-bind** is used with **class** and **style**. In addition to strings, the expressions can also evaluate to objects or arrays.

Class bindings

```
const isActive = ref(true)
const hasError = ref(false)
```

```
<div
  class="static"
  :class="{ active: isActive, 'text-danger': hasError }"
></div>
```

Renders

```
<div class="static active"></div>
```

Style bindings

```
const styleObject = reactive({
  color: 'red',
  fontSize: '13px'
})
```

```
<div :style="styleObject"></div>
```

```
<div style="font-size: 13px; color: red;"></div>
```

Vue.js

Conditional Rendering

Conditional Rendering in Vue makes it easy to toggle the presence of any element in the DOM based on a certain condition. The directives v-if and v-else are used for this purpose.

v-if/ v-else/ v-else-if

```
<div v-if="type === 'A'">
  A
</div>
<div v-else-if="type === 'B'">
  B
</div>
<div v-else-if="type === 'C'">
  C
</div>
<div v-else>
  Not A/B/C
</div>
```

v-show

```
<h1 v-show="ok">Hello!</h1>
```

The difference between **v-if** and **v-show** is that an element with **v-show** will always be rendered and remain in the DOM; **v-show** only toggles the display CSS property of the element.

Vue.js

List Rendering

List rendering is one of the most commonly used practices in front-end web development. Dynamic list rendering is often used to present a series of similarly grouped information in a concise and friendly format to the user. In almost every web application we use, we can see lists of content in numerous areas of the app.

We can use the v-for directive to render a list of items based on an array. The v-for directive requires a special syntax in the form of item in items, where items is the source data array and item is an **alias** for the array element being iterated on:

v-for

```
const items = ref([{ message: 'Foo' }, { message: 'Bar' }])
```

```
<li v-for="item in items">
  {{ item.message }}
</li>
```

Vue.js

Declaring Emitted Events

Emitted events can be explicitly declared on the component via the **defineEmits()** macro

```
<script setup>
const emit = defineEmits(['inFocus', 'submit'])
</script>
```

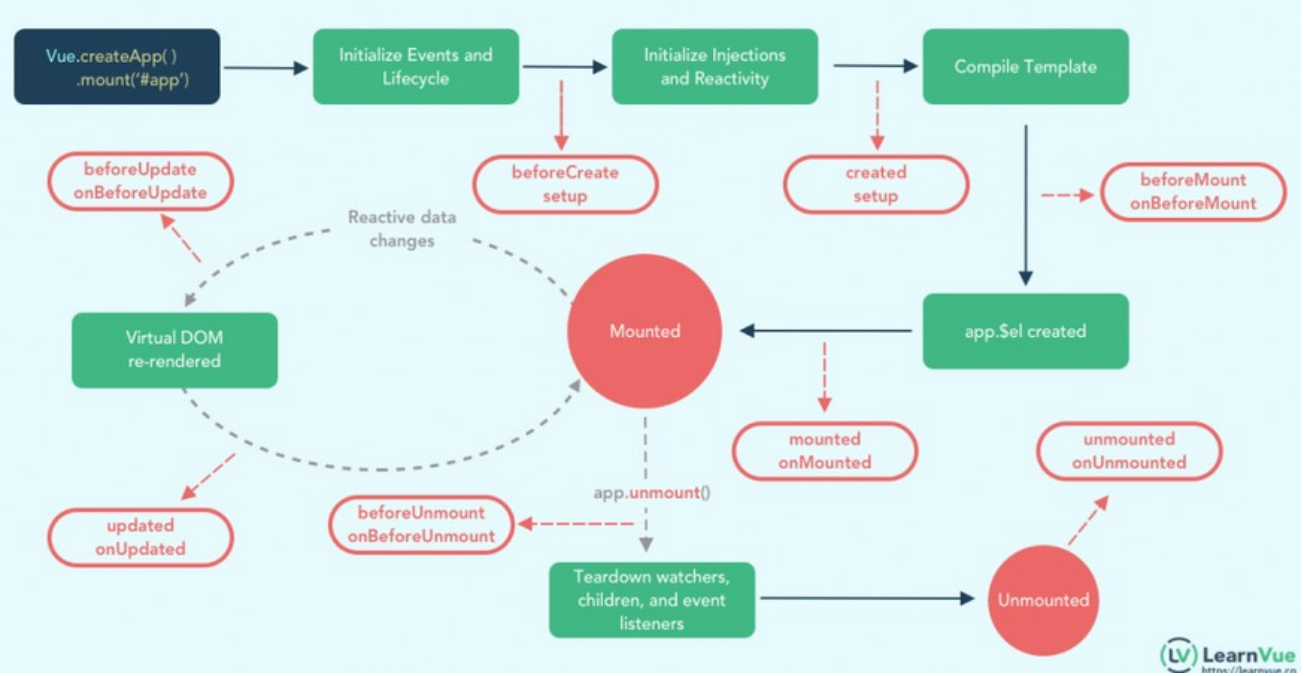
MINAB IT SOLUTIONS

Hahu Jobs

Vue.js

lifecycle hooks

Each Vue component instance goes through a series of initialization steps when it's created - for example, it needs to set up data observation, compile the template, mount the instance to the DOM, and update the DOM when data changes. Along the way, it also runs functions called lifecycle hooks, giving users the opportunity to add their own code at specific stages.



- **onBeforeMount** – called before mounting begins
- **onMounted** – called when component is mounted
- **onBeforeUpdate** – called when reactive data changes and before re-render
- **onUpdated** – called after re-render
- **onBeforeUnmount** – called before the Vue instance is destroyed
- **onUnmounted** – called after the instance is destroyed
- **onActivated** – called when a kept-alive component is activated
- **onDeactivated** – called when a kept-alive component is deactivated
- **onErrorCaptured** – called when an error is captured from a child component

Vue.js

Watchers

Computed properties allow us to declaratively compute derived values. However, there are cases where we need to perform "side effects" in reaction to state changes - for example, mutating the DOM, or changing another piece of state based on the result of

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

const question = ref('')
const answer = ref('Questions usually contain a question mark. ;-)')

// watch works directly on a ref
watch(question, async (newQuestion, oldQuestion) => {
  if (newQuestion.indexOf('?') > -1) {
    answer.value = 'Thinking...'
    try {
      const res = await fetch('https://yesno.wtf/api')
      answer.value = (await res.json()).answer
    } catch (error) {
      answer.value = 'Error! Could not reach the API. ' + error
    }
  }
})
</script>

<template>
  <p>
    Ask a yes/no question:
    <input v-model="question" />
  </p>
  <p>{{ answer }}</p>
</template>
```

Vue.js

Template Refs

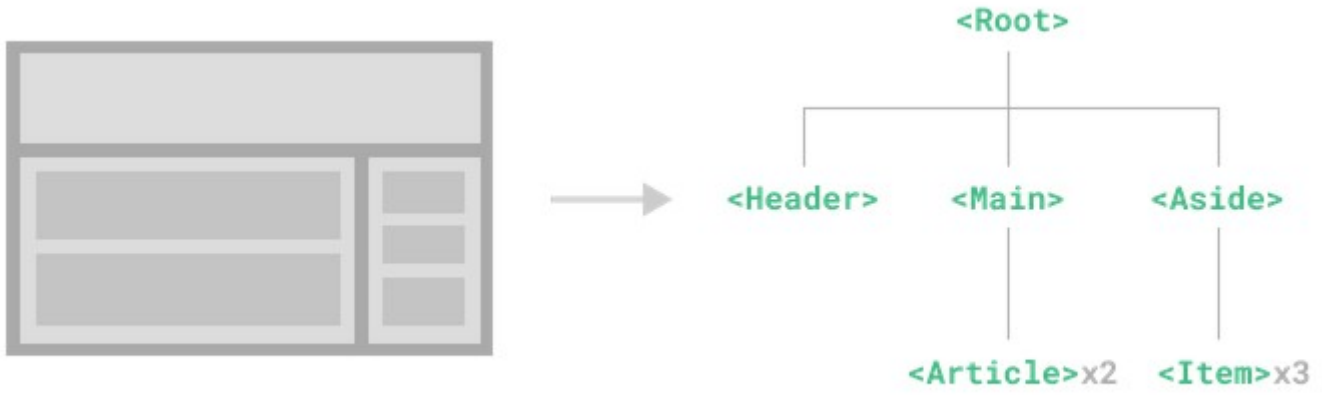
While Vue's declarative rendering model abstracts away most of the direct DOM operations for you, there may still be cases where we need direct access to the underlying DOM elements. To achieve this, we can use the special ref attribute:

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

// declare a ref to hold the element reference
// the name must match template ref value
const input = ref(null)

onMounted(() => {
  input.value.focus()
})
</script>

<template>
  <input ref="input" />
</template>
```

Vue.js

Components Basic

Components allow us to split the UI into independent and reusable pieces, and think about each piece in isolation. It's common for an app to be organized into a tree of nested



Vue.js

Components Basic

Components allow us to split the UI into independent and reusable pieces, and think about each piece in isolation. It's common for an app to be organized into a tree of nested components:

Defining a Component

we typically define each Vue component in a dedicated file using the .vue extension - known as a Single-File Component (SFC for short)

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

const count = ref(0)
</script>

<template>
  <button @click="count++">You clicked me {{ count }} times.</button>
</template>
```

MINAB IT SOLUTIONS

HahuJobs

Vue.js

Components Basic

Components allow us to split the UI into independent and reusable pieces, and think about each piece in isolation. It's common for an app to be organized into a tree of nested components:

Passing Props

Props are custom attributes you can register on a component. To pass a title to our blog post component, we must declare it in the list of props this component accepts, using the defineProps macro:

```vue
<!-- BlogPost.vue -->
<script setup>
defineProps(['title'])
</script>

<template>
  <h4>{{ title }}</h4>
</template>
```

```vue
<BlogPost title="My journey with Vue" />
<BlogPost title="Blogging with Vue" />
<BlogPost title="Why Vue is so fun" />
```

Vue.js

Components Basic

Components allow us to split the UI into independent and reusable pieces, and think about each piece in isolation. It's common for an app to be organized into a tree of nested components:

Listening to Events

some features may require communicating back up to the parent.

```
<template>
  <div class="blog-post">
    <h4>{{ title }}</h4>
    <button @click="$emit('enlarge-text')">Enlarge text</button>
  </div>
</template>
```

```
<BlogPost
  ...
  @enlarge-text="postFontSize += 0.1"
/>
```

MINAB IT SOLUTIONS

HahuJobs

Vue.js

Components Basic

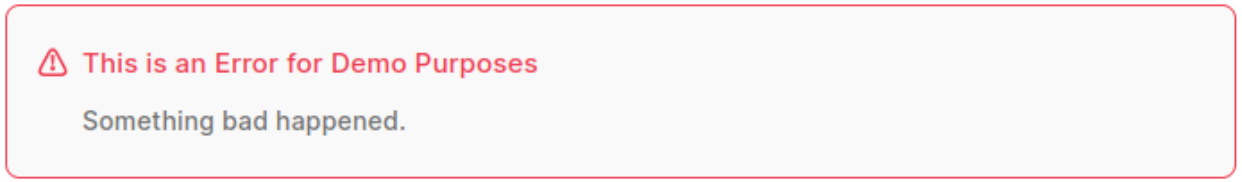
Components allow us to split the UI into independent and reusable pieces, and think about each piece in isolation. It's common for an app to be organized into a tree of nested components:

Content Distribution with Slots

Just like with HTML elements, it's often useful to be able to pass content to a component, like this:

```
<template>
  <div class="alert-box">
    <strong>This is an Error for Demo Purposes</strong>
    <slot />
  </div>
</template>

<style scoped>
.alert-box {
  /* ... */
}
</style>
```

```
<AlertBox>
  Something bad happened.
</AlertBox>
```



MINAB IT SOLUTIONS

HahuJobs

Vue.js

Setting up vue

We've already setup vue on previous class with tailwind css.
One might ask what **vite** is?

Vite is a build tool that aims to provide a faster and leaner development experience for modern web projects. It consists of two major parts:

- A dev server that provides rich feature enhancements over native ES modules, for example extremely fast Hot Module Replacement (HMR).
- A build command that bundles your code with Rollup, pre-configured to output highly optimized static assets for production.

MINAB IT SOLUTIONS

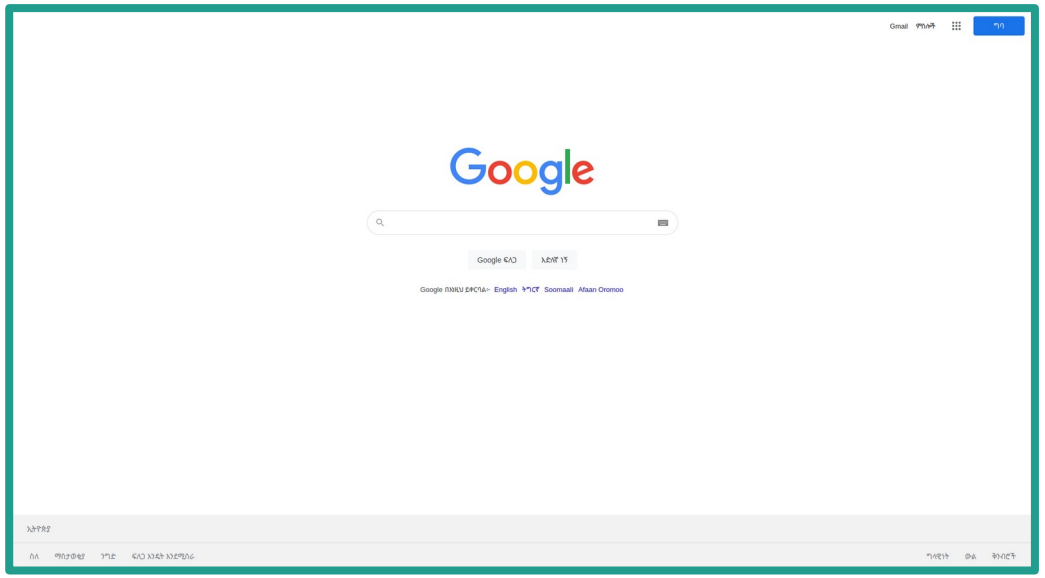
HahuJobs

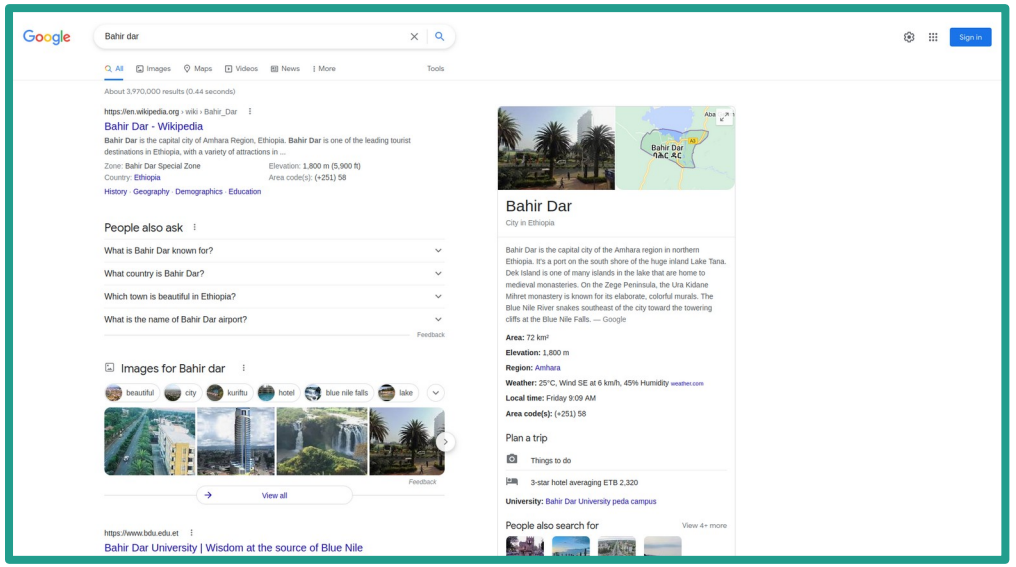
Vue.js

Hands on vue

We’ll extend what we’ve built on tailwind project with vue features

Convert those pages into smaller components like navbar, footer...





MINAB
IT SOLUTIONS

Thank you!

Hahu Jobs

ለሀገር ልጅ በሀገር ልጅ !

Michael Sahlu
michael.sahlu@hahu.jobs