Vue.js introduction

WASA: Web and Software Architecture

Enrico Bassetti

(JavaScript) Frameworks

What is a Framework

- More than libraries
- Inversion of control
- Extensible
- Non-user-modifiable (as opposed to templates)

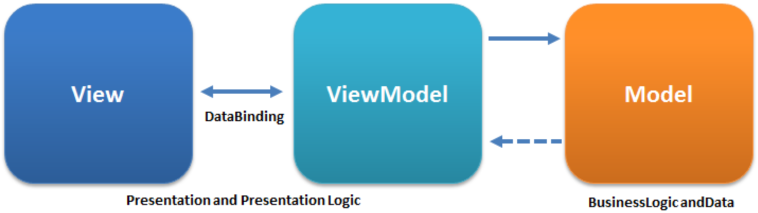
Why frameworks?

- JavaScript was not designed for developing applications
- Even if JS is being extended, some things can’t be part of it
- A lot of “boilerplate”

Common JavaScript frameworks

- React
- Angular
- Vue.js
- Ember.js
- GWT
- (many others, including the one invented a few minutes ago)

Web frameworks: Model-view-binder



Credits by Ugaya40 (Wikipedia) - CC BY-SA 3.0

TypeScript

Some JavaScript frameworks support TypeScript, a syntactical superset of JavaScript.

We won’t use TypeScript, but you may want to look at it if you pursue front-end development.

How front-end development works

1. Write JavaScript/HTML/CSS using framework rules/syntax
2. “Compile” it by using framework tools
3. Publish the result for testing/general use

Vue.js

What is Vue.js

From the Vue.js website:

*Vue (pronounced /vju:/, like view) 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 they simple or complex.*

Core features

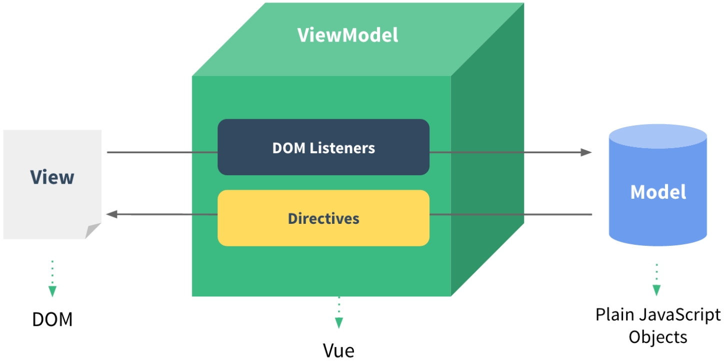
Core features:

- Declarative Rendering
- Reactivity

Way of using Vue.js

- Standalone (library-like)
- Embedded web components (embed Vue.js in existing legacy web apps)
- SPA, Single-Page Application
    - runs on the client
- Full-stack / Server-side rendering
    - runs on the server and client
- Static site generation
    - builds a static website
    - no code running on servers/clients

Model-view-binder



(C) by Vue.js documentation

API style: Options

Two possible styles:

- **Options API** (beginner-friendly)
- Composition API

Minimal example (JS part)

```
import { createApp } from 'vue'

createApp({
  data() {
    return {
      count: 0
    }
  }
}).mount('#app')
```

Minimal example (Template)

```html
<div id="app">
  <button @click="count++">
    Count is: {{ count }}
  </button>
</div>
```

Single-file components

A single-file component (SFC) is an HTML-like file that contains:

- JavaScript
- HTML (template)
- CSS

An SFC allows you to reuse components.

SFC example: extending the standard HTML button with custom behavior.

Single-file components (example)

```
<script>
export default {
  data() {
    return { count: 0 }
  }
}
</script>

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

<style scoped>
button { font-weight: bold; }
</style>
```

Vue.js playground

https://sfc.vuejs.org/

Anatomy of a Vue.js SPA

Index page

index.html contains the HTML for the single page. It may have non-Vue code, like external/custom JS/CSS.

Example:

```html
<!DOCTYPE html>
<html lang="en">
<head>
  <title>Example app</title>
  <meta charset="UTF-8">
</head>
<body>
<div id="app"></div>
<script type="module" src="/src/main.js"></script>
</body>
</html>
```

Main JS

In *`main.js`* we initialize Vue.js.

```
import {createApp, reactive} from 'vue'
import router from './router'
import App from './App.vue'
import Component1 from './components/Component1.vue'

const app = createApp(App)
app.component("Component1", Component1);
app.use(router)
app.mount('#app')
```

Main Vue.js component

In *`App.vue`* we have the main component of the SPA.

```
<script setup>
import { RouterView } from 'vue-router'
</script>
<script>
export default {}
</script>

<template>
  <header>App header</header>
  <main><RouterView /></main>
</template>

<style>
</style>
```

Router component

The router manages the routes for views inside the app.

```
import {createRouter, createWebHashHistory} from 'vue-router'
import HomeView from '../views/HomeView.vue'
import Page1View from '../views/Page1View.vue'
import Page2View from '../views/Page2View.vue'

const router = createRouter({
  history: createWebHashHistory(),
  routes: [
    {path: '/', component: HomeView},
    {path: '/link1', component: Page1View},
    {path: '/some/:id/link', component: Page2View},
  ]
});
export default router
```

Router

The previous code creates a “router” that you can manage by changing URLs. Each URL is associated with a different view of the app.

The actual link for *`/link1`* will be something like *`https://example.com/#/link1`*.