VueJS

The Progressive Library

Training Agenda

- ❑VueJS : Introduction
- ❑Vue Building Blocks
  - Events
  - Components
  - Directive
  - Mixins
  - FIlters
- ❑Data Binding
- ❑Remote Communication
- ❑Routing
- ❑State Management

Why VueJS?

- ES2015+ features
- Virtual DOM
- Performance & SPA
- Project architecture and Maintenance
- Component based architecture

What is VueJS?

VueJS is a progressive framework for building user interfaces (UI).

Super Features

- Data Binding
- XHR Calls
- Single Page Apps
- Templates
- Power of built-in Directives
- Component Reusability
- Virtual DOM
- Animation
- CLI tool
- Modularity

Directives

- Directives are instruction for VueJS to do things in a certain way.

Built-in Directives

v-text

- Updates the element’s

v-html

- Updates the element’s

v-show

- Toggles the element’s CSS property based on the truthy-ness of the expression value.

v-if

- Conditionally render the element based on the truthy-ness of the expression value.

V-else

- previous sibling element must have v-if or v-else-if.

V-else-if

- previous sibling element must have v-if or v-else-if.

Built-in Directives (Cntd...)

v-for

- Render the element or template block multiple times based on the source data.

v-on

- Attaches an event listener to the element.

v-bind

- Dynamically bind one or more attributes, or a component prop to an expression.

v-model

- Create a two-way binding on a form input element or a component.

v-slot

- Denote named slots or slots that expect to receive props.

v-once

- Render the element and component once only.

Computed Properties :

- Computed properties are like methods but with some differences.
- Computed Properties depends on normal properties. It will update any bindings that depend on computed property when normal property changes.
- Use a computed property for any complex logic.
- Computed properties are cached based on their reactive dependencies.
- Computed properties are by default getter-only, but you can also provide a setter when you need it.

Listening to Events

- Use the **v-on** directive to listen to DOM events and run some JavaScript when they're triggered.
- Access the original DOM event in using the special **$event** variable.

Event Modifiers :

.stop
.prevent
.capture
.self
.once
.passive

Key Modifiers :

.enter
.tab
.delete (captures both "Delete" and "Backspace" keys)
.esc
.space
.up
.down
.left
.right

Components

- A component controls a patch of screen real estate that we could call a view, and declares reusable UI building blocks for an application.
- Allows us to build large-scale applications composed of small, self-contained, and reusable components.
- Component Registration :
  - Local
  - Global
- Passing data into components -
  - Props (One-Way Data Flow)
  - Emitting Events
  - Callbacks
  - Event Bus

beforeCreate	- Called synchronously immediately after the instance has been initialized
created	- Called synchronously after the instance is created.
beforeMount	- Called right before the mounting begins
mounted	- Called after the instance has been mounted
beforeUpdate	- Called when data changes, before the DOM is patched
updated	- Called after a data change causes the virtual DOM to be re-rendered and patched.
activated	- Called when a kept-alive component is activated.
deactivated	- Called when a kept-alive component is deactivated.
beforeDestroy	- Called right before a Vue instance is destroyed. At this stage the instance is still fully functional.
destroyed	- Called after a Vue instance has been destroyed.

Life Cycle Diagram

Life Cycle Methods

Built-in Components

component	• A “meta component” for rendering dynamic components. • The actual component to render is determined by the prop.
transition	• Serve as transition effects for single element/component. • The only applies the transition behavior to the wrapped content inside.
keep-alive	• When wrapped around a dynamic component, caches the inactive component instances without destroying them.
slot	• Serve as content distribution outlets in component templates. • Itself will be replaced.

Filters

- Vue.js allows you to define filters that can be used to apply common text formatting.
- Filters should be appended to the end of JavaScript expression, denoted by the “pipe” symbol.
- Filters are usable in two places:
  - mustache interpolations
  - v-bind expressions

Mixins

- Mixins share reusable code among components.
- When a component uses mixin, all options of mixin become a part of the component options.
- The mixins option accepts an array of mixin objects.
- Mixin hooks are called in the order they are provided, and called before the component’s own hooks.

Vue-resource

The plugin for Vue.js provides services for making web requests and handle responses using a XHR or JSONP.

Supports the Promise API and URI Templates

Supports interceptors for request and response

Supports latest Firefox, Chrome, Safari, Opera and IE9+

```
npm install vue-resource
```

vue-resource Documentation

vue-router (build SPA)

Vue-router Features	Nested route/view mapping
	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, with auto-fallback in IE9
	Customizable Scroll Behavior

Vue-router Documentation

Routing

Routing allows to:

- Maintain the state of the application
- Implement modular applications
- Implement the application based on the roles (certain roles have access to certain URLs)

5 steps routing:

- Configuring routes with components
- Tell Vue about routing app
- Setting up the routing links
- Provide space on template to load the component

Routing (Cntd..)

Child routing

(using children property)

Programmatic navigation

(using $router)

Routes with parameters

(using $route.params)

Query Parameters

(using $route.query)

Route guard

(beforeEach, beforeEnter, beforeRouteEnter)

Vuex
(Centralized State Management for Vue.js)

- Vuex is a state management pattern + library for Vue.js applications.
- serves as a centralized store for all the components in an application, with rules ensuring that the state can only be mutated in a predictable fashion.

vuex Documentation

vuex : Store

At the center of every Vuex application is the store.

A "store" is basically a container that holds your application state.

Vuex stores are reactive. When Vue components retrieve state from it, they will reactively and efficiently update if the store's state changes

We cannot directly mutate the store's state. The only way to change a store's state is by explicitly committing mutations. This ensures every state change leaves a track-able record, and enables tooling that helps us better understand our applications

vuex : State

Vuex uses a single state tree

The single object contains all application level state and serves as the "single source of truth".

Usually we will have only one store for each application.

A single state tree makes it straightforward to locate a specific piece of state, and allows us to easily take snapshots of the current app state for debugging purposes.

mapState helper which generates computed getter functions for us.

vuex : Getters

Vuex allows us to define "getters" in the store.

We can think of them as computed properties for stores.

Like computed properties, a getter's result is cached based on its dependencies, and will only re-evaluate when some of its dependencies have changed.

Getters will receive the state as their 1st argument.

The getters will be exposed on the **store.getters** object, and you access values as properties.

The **mapGetters** helper simply maps store getters to local computed properties.

vuex : Mutations

The only way to actually change state in a Vuex store is by committing a *mutation*.

Vuex mutations are very similar to events: each mutation has a string *type* and a *handler*.

The *handler* function is where we perform actual state modifications, and it will receive the state as the first argument.

To invoke a mutation handler, you need to call **store.commit** with its *type*.

We can pass an additional argument to **store.commit**, which is called the *payload* for the mutation.

Mutations must be synchronous.

vuex : Actions

- Actions are similar to mutations, the differences being that:
  - Instead of mutating the state, actions commit mutations.
  - Actions can contain arbitrary asynchronous operations.
- Action handlers receive a context object which exposes the same set of methods/properties on the store instance.
- Actions are triggered with the **store.dispatch** method.

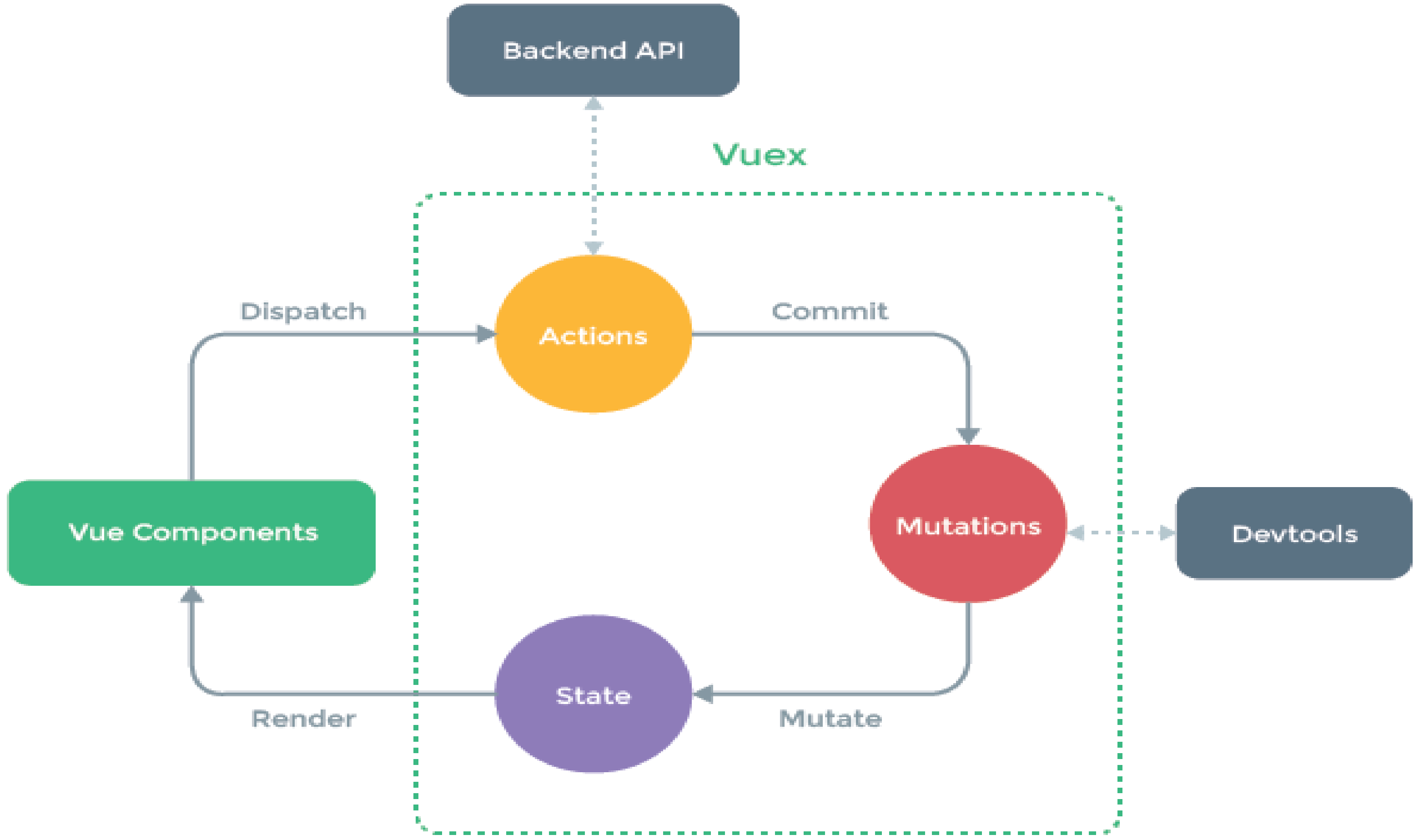
vuex : Modules

Due to using a single state tree, all state of our application is contained inside one big object. However, as our application grows in scale, the store can get really bloated.

To help with that, Vuex allows us to divide our store into modules.

Each module can contain its own state, mutations, actions, getters, and even nested modules.

Vuex : Centralized State Management

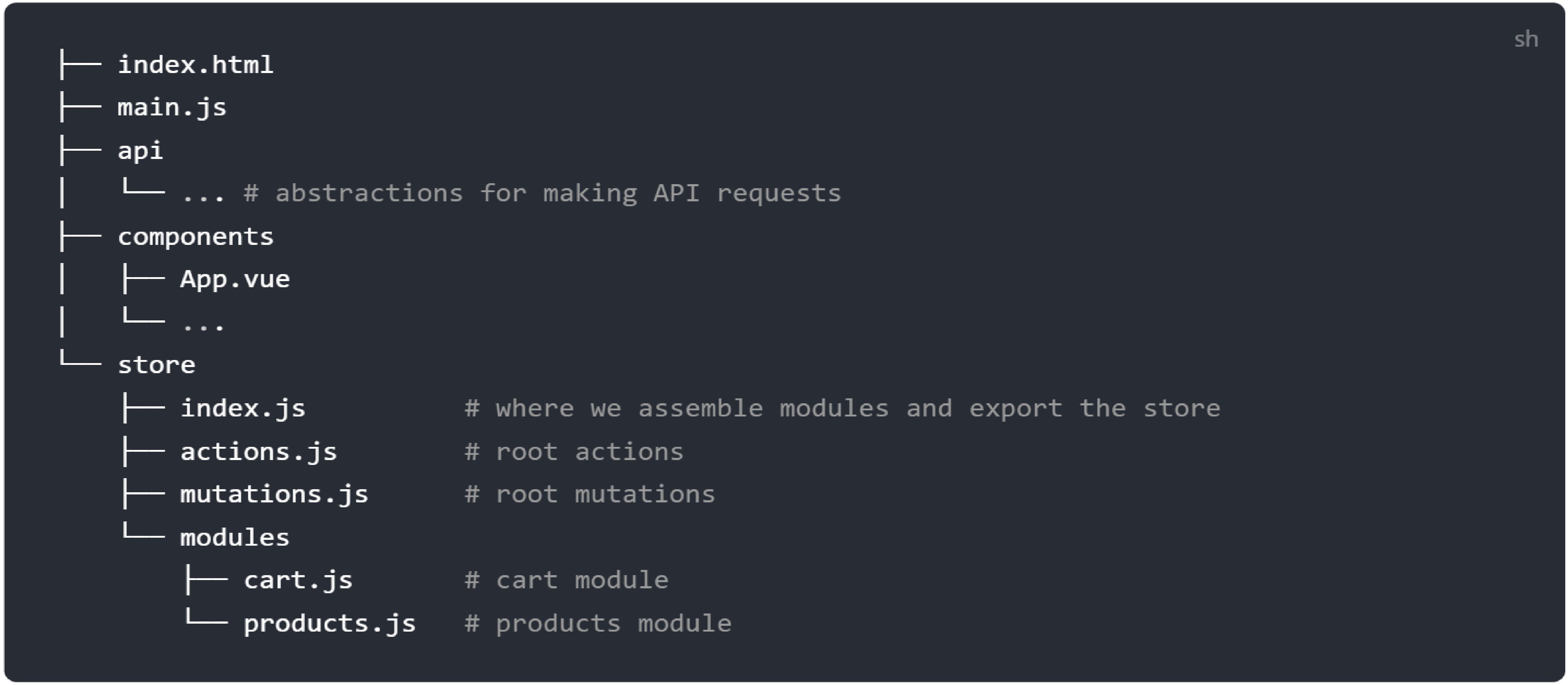


Application Structure

- Vuex doesn't really restrict how you structure your code. Rather, it enforces a set of high-level principles:
  - Application-level state is centralized in the store.
  - The only way to mutate the state is by committing mutations, which are synchronous transactions.
  - Asynchronous logic should be encapsulated in, and can be composed with actions.

- As long as you follow these rules, it's up to you how to structure your project. If your store file gets too big, simply start splitting the actions, mutations and getters into separate files.

Sample Project Structure for any Non-trivial App



```sh
├── index.html
├── main.js
├── api
│   └── ... # abstractions for making API requests
├── components
│   ├── App.vue
│   └── ...
└── store
    ├── index.js          # where we assemble modules and export the store
    ├── actions.js        # root actions
    ├── mutations.js      # root mutations
    └── modules
        ├── cart.js       # cart module
        └── products.js   # products module
```

Debugging Vue Apps

- Console Debugging
- VueJS devtools Chrome Plugin
- Chrome Debugging tool
- Debugger Statement

References

Books

Learning Vue.js (Packt publication)

Web

https://vuejs.org/

https://www.npmjs.com/package/vue-resource

https://github.com/pagekit/vue-resource

https://vuejs.org/v2/guide/routing.html

https://router.vuejs.org/

https://vuex.vuejs.org/

https://github.com/vuejs/vuex