



Web Applications Powered by VueJS!

From Concept to Launch!



Web Applications Powered by VueJS!

From Concept to Launch!

Presented by
Peter Pavlovich
pavlovich@gmail.com

Introductions ...

- Peter Pavlovich

- Front-End Evangelist and Technology Addict

- CTO, Censinet, Inc. (Supply Chain Risk Management for the Healthcare Sector)

- Vue/Rails/Python, Postgres/MongoDB, AWS/Heroku

- Principal Advisor, Embue, Inc. (Smart Apartment Management Systems)

- Angular/Vue/Meteor, Node, Scala/Java, Postgres/MongoDB, AWS

Introductions ...

- Peter Pavlovich
 - Entrepreneur, Open Source Contributor and Community Member
 - Instructor, Mentor and Speaker.
 - <http://linkedin.com/in/peterpavlovich>
 - pavlovich@gmail.com
 - @ppavlovich

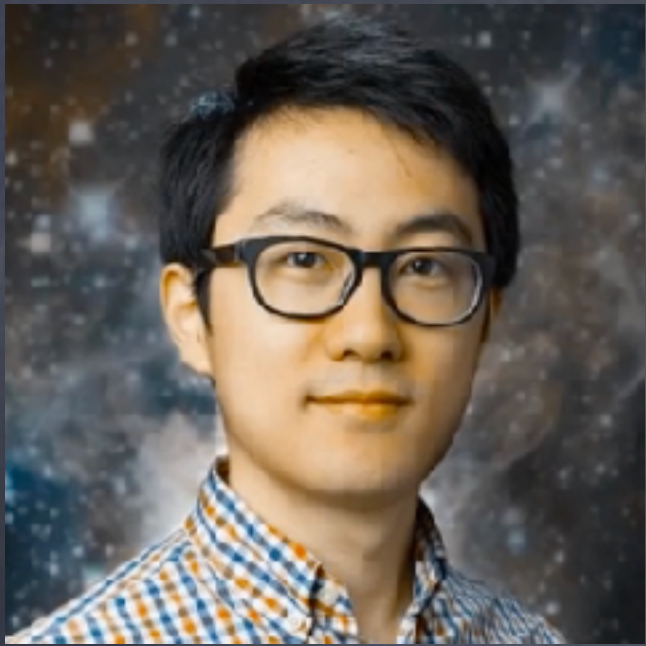
Plan for our session ...

- Build and deploy a cool Task Manager App
- Along the way we will:
 - Explore all the major features of VueJS
 - Understand the add-on technologies
 - Build a Web app
 - Utilize Vuex, Vuetify and Nuxt
 - Witness amazingly easy application development
 - Love how simple, powerful, and fun VueJS is!

Local Network Server

- network: nfjs_salon_1
- <http://10.0.1.2/angular>

What is VueJS?



What is VueJS

- A web application, front-end framework
- Based on work by Evan You, an ex-googler from the Angular world.
- "I figured, what if I could just extract the part that I really liked about Angular and build something really lightweight"

Why VueJS?

- It is:
 - Approachable
 - Versatile
 - Performant
 - Maintainable
 - Testable

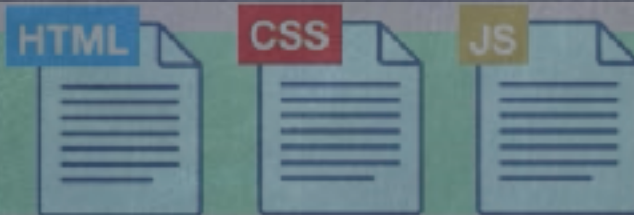
Why VueJS?

- Progressive
 - Convert your existing app over time
 - Coexists with:
 - Angular, React, Polymer, etc.
 - Java/Grails, .net, Ruby/Rails, Meteor

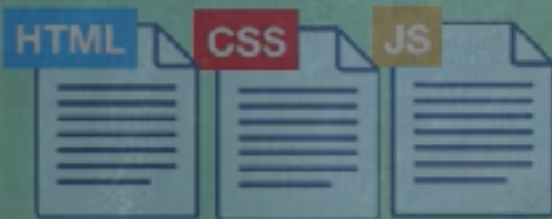


Why VueJS?

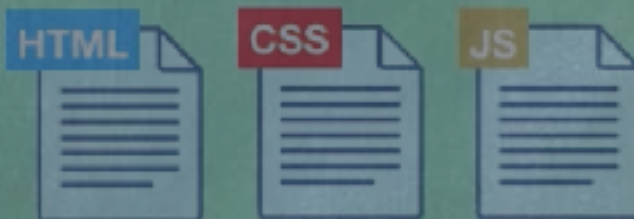
Header & Navigation



Product Image



Product Description



Similar Products



- Component Based

- Stateless, functional components for performance

- Stateful components for flexibility/power

Why VueJS?

- Simultaneous support for:
 - Multiple scripting languages
 - ES5, ES6, ES7, coffeescript, typescript
 - Multiple styling libraries
 - Plain CSS, Less, Sass, Stylus
 - Multiple templating languages
 - HTML, Handlebars, JSX, Pug/Jade

Vue is Reactive!

- When the data for our app changes the UI will update to reflect those changes.

Reactive demo

- Build a simple Vue page and see it react!

Why another
framework?

Vue vs React

- Both: Use a virtual DOM
- Both: Components are reactive & composable
- Both: Well factored:
 - Core library
 - Routing, global state in separate libraries
- Both: Perform well with little differences

Vue vs React

- When React re-renders a component, it rerenders the entire component subtree.
 - Need 'shouldComponentUpdate' a lot = extra work for developers
- Vue does this for you automatically without extra coding for the developer.

Vue vs React

- In React, everything is just javascript
 - JSX templates everywhere
 - CSS is also 'javascript'
- Vue supports JSX when you need that but also supports plain HTML templates when you don't. Also supports Jade/Pug, handlebars, ...

Vue vs React

- Scaling
 - Big learning curve with React:
 - JSX, ES6+, build systems, etc.
 - Vue is simpler, more flexible, less opinionated.

Vue vs Angular

- AngularJS

- Vue is simpler, component based, faster.

- Angular (2+)

- Vue is smaller (50% less), less opinionated/flexible and does not mandate the use of Typescript.

- Vue has a much lower learning curve.

VueJS Ecosystem

- Lots of great plugins:
 - Vuex: Redux for vue
 - Vue-Router: Full featured Router for vue
 - Nuxt: developer framework for vue
 - vuexfire/vuexFire: Firebase bindings for vue
 - Vuemotion: Easy state transitions for vue

VueJS Ecosystem

- More great plugins:
 - axios: easy REST calls integrated with vue
 - Vuelidate: model validation framework
 - Vue-apollo: Apollo/GraphQL integration
 - Vue-i18n: Internationalization plugin
 - vuetify: Material component framework

VueJS Ecosystem

- Even more great plugins:

- https://curated.vuejs.org/module/github_com::vuetifyjs

Under the hood

How does Vue Work?

- In JS, you create an instance of Vue.
- In the constructor, you pass an options obj.
- You identify the DOM element from your HTML that you wish to be controlled by Vue.
- Vue takes the HTML from that element and creates a template function from it.
- The template function generates the HTML for the browser using info/data from the vue instance.

Creating Vue instance

```
new Vue({  
  el: '#app'  
});
```

```
<div id="app">  
  <span>I love ice cream!</span>  
</div>
```


Creating Vue instance

```
new Vue({  
  el: '#app',  
  data: {  
    product: 'ice cream'  
  },  
  ...  
});
```

Using dynamic data

```
new Vue({  
  el: '#app',  
  data: {  
    product: 'ice cream'  
  }  
});
```

```
<div id="app">  
  <span>I love {{ product }}!</span>  
</div>
```


Using methods

```
new Vue({  
  el: '#app',  
  methods: {  
    locateMe: function() {  
      return "Denver"  
    }  
  }  
});
```

```
<div id="app">  
  <span>I love it in {{ locateMe() }}!</span>  
</div>
```

Using data & methods

```
new Vue({  
  el: '#app',  
  data: { place: 'Denver' },  
  methods: {  
    locateMe: function() {  
      return this.place;  
    }  
  }  
});
```

```
<div id="app">  
  <span>I love it in {{ locateMe() }}!</span>  
</div>
```


Dynamic HTML attributes

```
new Vue({  
  el: '#app',  
  data: {  
    company: {  
      name: 'Censinet',  
      link: 'http://censinet.com'  
    }  
  }  
});
```

```
<div id="app">  
  <span>I love working at  
    <a href="{{ company.link }}">{{ company.name }}</a>  
  </span>  
</div>
```

Vue's v-bind directive

```
new Vue({  
  el: '#app',  
  data: {  
    company: {  
      name: 'Censinet',  
      link: 'http://censinet.com'  
    }  
  }  
});
```

```
<div id="app">  
  <span>I love working at  
    <a v-bind:href="company.link">{{ company.name }}</a>  
  </span>  
</div>
```


Vue's binding ":" shortcut

```
new Vue({  
  el: '#app',  
  data: {  
    company: {  
      name: 'Censinet',  
      link: 'http://censinet.com'  
    }  
  }  
});
```

```
<div id="app">  
  <span>I love working at  
    <a :href="company.link">{{ company.name }}</a>  
  </span>  
</div>
```

Other directives: v-once

```
new Vue({  
  el: '#app',  
  data: {  
    food: 'cheesecake'  
  },  
  methods: {  
    updateName: function() {  
      this.food = 'peas'  
    }  
  }  
});
```

```
<div id="app">  
  <span v-once>I love {{ food }}</span>  
  <span>{{ food }} is my comfort food!</span>
```


Other directives: v-html

```
new Vue({  
  el: '#app',  
  data: {  
    favorite: {  
      name: 'Google',  
      link: '<a href="http://google.com">Google</a>`  
    }  
  }  
});
```

```
<div id="app">  
  <p>Try searching at {{favorite.name}}!</p>  
  <p>{{ favorite.link }}</p>  
  <p v-html="favorite.link"></p>  
</div>
```

Exercise

- Use <http://codepen.io> or <http://jsfiddle.net>
- Import <https://unpkg.com/vue/dist/vue.js>
- Create a hard-coded task with id (number), name, icon URL (find a suitable one hosted on the web) and owner (email) and store that data in a vue instance's data.
- Output the task icon, name and priority information.
- Also, show the current date and time retrieved from the vue instance using a dynamic method.

Other directives: v-for (collection)

```
new Vue({  
  el: '#app',  
  data: {  
    foods: [  
      'rice', 'apples', 'steak'  
    ]  
  })
```

```
<div id="app">  
  <ul>  
    <li v-for="food in foods">{{ food }}</li>  
  </ul>  
</div>
```


Other directives:

v-for (collection/index)

```
new Vue({  
  el: '#app',  
  data: {  
    foods: [  
      'rice', 'apples', 'steak'  
    ]  
  })
```

```
<div id="app">  
  <ul>  
    <li v-for="(f, i) in foods">({{i}}) {{f}}</li>  
  </ul>  
</div>
```

Other directives:

v-for (object/with key)

```
new Vue({  
  el: '#app',  
  data: {  
    foods: {  
      rice: {type: 'grains', size: 'cup'},  
      apple: {type: 'fruit', size: 'each'}  
    }  
  }  
});
```

```
<div id="app">  
  <ul> <li v-for="(p, name) in foods">  
    {{ `${name} (${p.type}) is sold by the ${p.size}` }}  
  </li> </ul>  
</div>
```

Other directives: v-for (over numbers)

```
new Vue({  
  el: '#app'  
});
```

```
<div id="app">  
  <ul>  
    <li v-for="num in 100">  
      The count is now {{ num }}  
    </li>  
  </ul>  
</div>
```


Other directives: v-on

```
new Vue({
  el: '#app',
  data: {
    foods: ['rice', 'apples', 'steak']
  },
  methods: {
    enter: (event) => {event.target.style.backgroundColor: 'red' }
    exit: (event) => {event.target.style.backgroundColor: 'transparent' }
  }) ;
```

```
<div id="app">
  <ul>
    <li v-for="food in foods"
      v-on:mouseenter="enter"
      v-on:mouseleave="exit">{{ food }}</li>
  </ul>
</div>
```

Vue's listen "@" shortcut

```
new Vue({
  el: '#app',
  data: {
    foods: ['rice', 'apples', 'steak']
  },
  methods: {
    enter: (event) => {event.target.style.backgroundColor: 'red' }
    exit: (event) => {event.target.style.backgroundColor: 'transparent' }
  }) ;
```

```
<div id="app">
  <ul>
    <li v-for="food in foods"
      @mouseenter="enter"
      @mouseleave="exit">{{ food }}</li>
  </ul>
</div>
```

Passing arguments

```
new Vue({
  el: '#app',
  data: {
    foods: ['rice', 'apples', 'steak']
  },
  methods: {
    enter: (food, event) => {event.target.style.backgroundColor: 'red' }
    exit: (food, event) => {event.target.style.backgroundColor:
      'transparent' }
  }) ;
```

```
<div id="app">
  <ul>
    <li v-for="food in foods"
      @mouseenter="enter(food, $event)"
      @mouseleave="exit(food, $event)">{{ food }}</li>
  </ul>
</div>
```


Exercise

- Hardcode a list of tasks in a vue instance
- Display that list using an unordered list showing only the name
- When you click on one of these items, change its background to indicate it is selected and display the details of it below the task list.
- If you click it a second time, 'unselect' it and clear the area below the task list.

Using modifiers

```
const myApp = new Vue({
  el: '#app',
  data: { x: 0, y: 0 },
  methods: {
    move: function(e){ this.x = e.clientX; this.y = e.clientY; },
    noMove: function(event){ event.stopPropagation(); }
  }
});

<div id="app">
  <div @mousemove="move">
    This is a test
    <span @mousemove="Nomove"> ... this is only a test!</span>
    <span @mousemove.stop> ... this, too, is only a test!</span>
  </div>
  <div> {{ x }} / {{ y }}</div>
</div>
```

Key event modifiers

```
const myApp = new Vue({
  el: '#app',
  data: { text: 'nothing yet' },
  methods: {
    key: function({target}){ this.text = target.value },
    clear: function({target}){
      target.value = "";
      this.text = 'cleared';
    }
  }
});

<div id="app">
  <div>
    <input type="text" @focus="clear" @keyup.enter.space="key" />
  </div>
  <div> {{ text }} </div>
</div>
```


Exercise

- Change the previous exercise so that when you select a task, an input field containing its name appears. The input field is contained in a form and is displayed below the list.
- When you change the text in the input field and press enter, the form is submitted and the edited text becomes the new name for the selected task and the task is then unselected.
- If you click a selected task before pressing enter, it simply is unselected with no update and the input field is hidden.

2-way binding: v-model

```
const myApp = new Vue({
  el: '#app',
  data: { text: 'nothing yet' },
  methods: {
    key: function({target}){ this.text = target.value },
    clear: function({target}){
      target.value = "";
      this.text = 'cleared';
    }
  }
});

<div id="app">
  <div>
    <input type="text" @focus="clear" @keyup.enter.space="key" />
  </div>
  <div> {{ text }} </div>
</div>
```

Computed properties

```
const myApp = new Vue({
  el: '#app',
  data: { text: 'nothing yet' },
  methods: {
    key: function({target}){ this.text = target.value },
    clear: function({target}){
      target.value = "";
      this.text = 'cleared';
    }
  }
});

<div id="app">
  <div>
    <input type="text" @focus="clear" @keyup.enter.space="key" />
  </div>
  <div> {{ text }} </div>
</div>
```


Watching properties

```
const myApp = new Vue({  
  el: '#app',  
  data: { text: 'initial' },  
  watch: {  
    text: function(value, original){ console.log(`My text has  
changed to ${value} from ${original}`) },  
  },  
  methods: { changeText: function(){this.text = 'updated'} }  
});
```

```
<div id="app">  
  <div>  
    <span>My text is {{ text }}</span>  
  </div>  
  <button @click="changeText">Update text</button>  
</div>
```

Exercise

- Convert the previous example to use v-model to update a temporary copy of the selected task prior to updating.
- Display a confirmation dialog to the user if they attempt to unselect or select a different task when editing a task and unsaved changes are present.

Dynamic CSS

```
const myApp = new Vue({  
  el: '#app',  
  data: { isRed: false },  
  methods: {  
    toggleColor: function() { this.isRed = !this.isRed }  
  }  
});
```

```
<div id="app">  
  <div class="demo-text"  
    @click="toggleColor"  
    :class="{ 'make-it-red': isRed, 'make-it-blue': !isRed }" >  
    Click to change colors  
  </div>  
</div>
```

css is as you would expect it to be.

Dynamic CSS (2)

```
const myApp = new Vue({  
  el: '#app',  
  data: { isRed: false },  
  methods: { toggleColor: function() { this.isRed = !this.isRed } },  
  computed: { myClasses: function() {  
    return { 'make-it-red': this.isRed, 'make-it-blue': !this.isRed }  
  }  
});
```

```
<div id="app">  
  <div class="demo-text"  
    @click="toggleColor"  
    :class="myClasses" >  
    Click to change colors  
  </div>  
</div>
```

css is as you would expect it to be.

Dynamic CSS (3)

```
const myApp = new Vue({  
  el: '#app',  
  data: { isRed: false, dynClass: 'super-bold' },  
  methods: { toggleColor: function() { this.isRed = !this.isRed } }  
});
```

```
<div id="app">  
  <div class="demo-text"  
    @click="toggleColor"  
    :class="[dynClass, {'make-it-red': isRed}]" >  
    Click to change colors  
  </div>  
</div>
```

css is as you would expect it to be.

Dynamic CSS (4)

```
const myApp = new Vue({  
  el: '#app',  
  data: { bgColor: 'green', bdrStyle: '1px solid blue' }  
});
```

```
<div id="app">  
  <div class="demo-text"  
    :style="{ 'background-color': bgColor, borderTop: bdrStyle}">  
    Sample text  
  </div>  
</div>
```

css is as you would expect it to be.

Conditional Rendering: v-if / v-else-if / v-else

```
const myApp = new Vue({  
  el: '#app',  
  data: { showFormal: false, showCasual: false }  
});
```

```
<div id="app">  
  <div v-if="showFormal">  
    Greetings!  
  </div>  
  <div v-else-if="showCasual">  
    Hi there!  
  </div>  
  <div v-else>  
    There you are!  
  </div>  
</div>
```

Conditional Rendering: v-show

```
const myApp = new Vue({  
  el: '#app',  
  data: { showFormal: false, showCasual: false }  
});
```

```
<div id="app">  
  <div v-show="showFormal">  
    Greetings!  
  </div>  
  <div v-show="showCasual">  
    Hi there!  
  </div>  
</div>
```

Rendering multiple top-level elements

```
const myApp = new Vue({  
  el: '#app',  
  data: { showMe: true }  
});
```

```
<div id="app">  
  <template v-if="showMe">  
    <div> One </div>  
    <div> Two </div>  
  </template>  
</div>
```


Referencing HTML elements

```
const myApp = new Vue({  
  el: '#app',  
  data: { showMe: true },  
  methods: {  
    updateText: function() {  
      this.$refs.one.innerText = 'New One'  
    }  
  }  
});
```

```
<div id="app">  
  <template v-if="showMe">  
    <div ref="one" @click="updateText"> One </div>  
    <div> Two </div>  
  </template>  
</div>
```

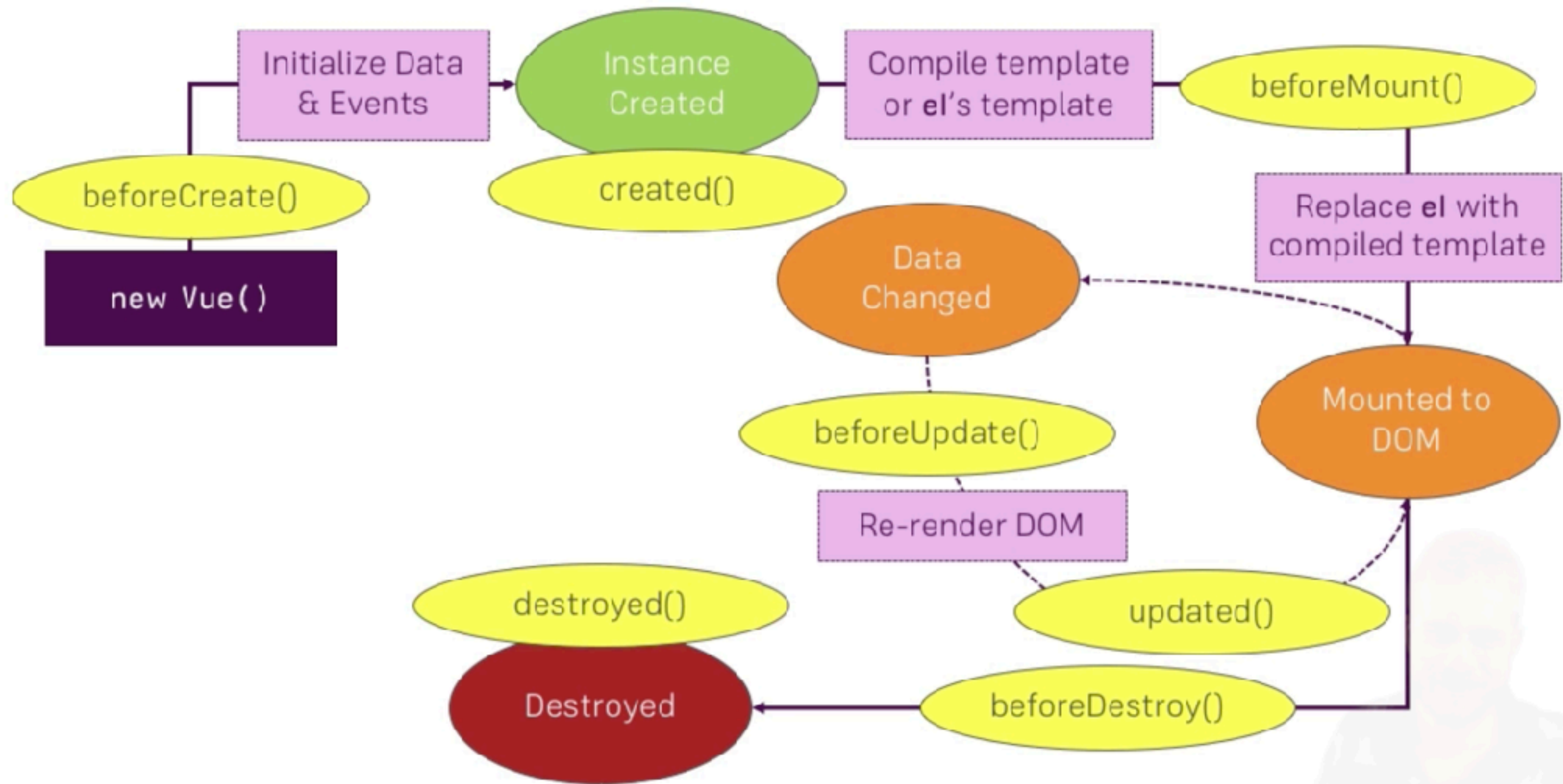
Manually Mounting Vue

```
const myApp = new Vue({
  data: { showMe: true },
  methods: {
    updateText: function() {
      this.$refs.one.innerText = 'New One'
    }
  }
});

myApp.$mount( '#app' );

<div id="app">
  <template v-if="showMe">
    <div ref="one" @click="updateText"> One </div>
    <div> Two </div>
  </template>
</div>
```

Vue instance lifecycle



Project Setup

Install WebStorm

- <https://www.jetbrains.com/webstorm/download/>

Install Sourcetree

- <https://www.atlassian.com/software/sourcetree>

Install the vue-cli

```
npm install -g yarn  
npm install -g @vue/cli
```

Using the vue-cli

```
> cd ~/projects  
projects> vue create taskmaster
```

Vue CLI v3.0.0-rc.4

? Please pick a preset:

censinet_standard (vue-router, vuex, sass, babel, typescript, pwa, eslint, unit-mocha, e2e-cypress)

Standard_presentation_config (babel, eslint)

NFJS_Workshop (vue-router, vuex, less, babel, eslint)

default (babel, eslint)

> Manually select features

Vue CLI v3.0.0-rc.4

? Please pick a preset: Manually select features

? Check the features needed for your project:

> ☒ Babel

☐ TypeScript

☐ Progressive Web App (PWA) Support

☐ Router

☐ Vuex

☒ CSS Pre-processors

☒ Linter / Formatter

☐ Unit Testing

☐ E2E Testing

Using the vue-cli

Vue CLI v3.0.0-rc.4

- ? Please pick a preset: Manually select features
- ? Check the features needed for your project: Babel, CSS Pre-processors, Linter
- ? Pick a CSS pre-processor (PostCSS, Autoprefixer and CSS Modules are supported by default): SCSS/SASS
 - > LESS
 - Stylus

Vue CLI v3.0.0-rc.4

- ? Please pick a preset: Manually select features
- ? Check the features needed for your project: Babel, CSS Pre-processors, Linter
- ? Pick a CSS pre-processor (PostCSS, Autoprefixer and CSS Modules are supported by default): LESS
- ? Pick a linter / formatter config:
 - ESLint with error prevention only
 - ESLint + Airbnb config
 - ESLint + Standard config
 - > ESLint + Prettier

Using the vue-cli

Vue CLI v3.0.0-rc.4

```
? Please pick a preset: Manually select features
? Check the features needed for your project: Babel, CSS Pre-processors, Linter
? Pick a CSS pre-processor (PostCSS, Autoprefixer and CSS Modules are supported by default): LESS
? Pick a linter / formatter config: Prettier
? Pick additional lint features: (Press <space> to select, <a> to toggle all, <i> to invert selection)
> ☒ Lint on save
  ☐ Lint and fix on commit
```

Vue CLI v3.0.0-rc.4

```
? Please pick a preset: Manually select features
? Check the features needed for your project: Babel, CSS Pre-processors, Linter
? Pick a CSS pre-processor (PostCSS, Autoprefixer and CSS Modules are supported by default): LESS
? Pick a linter / formatter config: Prettier
? Pick additional lint features: Lint on save
? Where do you prefer placing config for Babel, PostCSS, ESLint, etc.? (Use arrow keys)
> In dedicated config files
  In package.json
```

Using the vue-cli

Vue CLI v3.0.0-rc.4

```
? Please pick a preset: Manually select features
? Check the features needed for your project: Babel, CSS Pre-processors, Linter
? Pick a CSS pre-processor (PostCSS, Autoprefixer and CSS Modules are supported by default): LESS
? Pick a linter / formatter config: Prettier
? Pick additional lint features: Lint on save
? Where do you prefer placing config for Babel, PostCSS, ESLint, etc.? In dedicated config files
? Save this as a preset for future projects? (y/N) █
```

Vue CLI v3.0.0-rc.4

```
✖ Creating project in /Users/pavlovich/Documents/Presentations/vue/projects/taskmaster.hasProjectGit
✨ Creating project in /Users/pavlovich/Documents/Presentations/vue/projects/taskmaster.
🚧 Initializing git repository...
⚙ Installing CLI plugins. This might take a while...
```

yarn install v1.7.0

info No lockfile found.

[1/4] 🔍 Resolving packages...

[2/4] 📦 Fetching packages...

[3/4] 🔗 Linking dependencies...

success Saved lockfile.

✨ Done in 21.82s.

Using the vue-cli

```
🚀 Invoking generators...
📦 Installing additional dependencies...

yarn install v1.7.0
[1/4] 🔍 Resolving packages...
[2/4] 🚚 Fetching packages...
[3/4] 🔗 Linking dependencies...
[4/4] 🏗 Building fresh packages...

success Saved lockfile.
✨ Done in 4.54s.

📌 Running completion hooks...

📄 Generating README.md...

🎉 Successfully created project taskmaster.
👉 Get started with the following commands:

$ cd taskmaster
$ yarn serve

➔ projects █
```


[Home](#) | [About](#)



Welcome to Your Vue.js App

For guide and recipes on how to configure / customize this project,
check out the [vue-cli documentation](#).

Installed CLI Plugins

[babel](#) [eslint](#)

Essential Links

[Core Docs](#) [Forum](#) [Community Chat](#) [Twitter](#)

Ecosystem

[vue-router](#) [vuex](#) [vue-devtools](#) [vue-loader](#) [awesome-vue](#)

Exercise

- Create a new Vue project named 'taskmaster'
- Use the CLI
- Follow the selections as on the slides.

Review of base template

- Demo ...

Vue's ecosystem

- Vue has a number of 'plug-ins'
 - Official
 - Vue-router
 - Vuex (centralized store – like Redux)
 - Community
 - Vuetify (Material Design Inspired UI)

Let's add our UI widgets

```
projects/taskmaster> vue add vuetify
```

Review of Vuetify App

- Demo ...

Basic *.vue Component

```
<template lang="... | default is the Vue template syntax">  
  <SomeComponentINeed></SomeComponentINeed>  
</template>
```

```
<script lang="typescript | coffee | default is ES6">  
  import <SomeComponentINeed>;
```

```
  export default {  
    name: 'MyComponentName',  
    props: [propName1, propName2, ...],  
    data: () => { return {stateVar1: initValue1, ...} },  
    computed: {propName: () => {return value}, ...},  
    methods: {methodName: () => {... do something ...}},  
    watch: {propName: (newValue, oldValue){... do something ...}, ...}  
    lifecycleMethod1(){ ... do something ...}, ...,  
    components: {LocalCmpName: SomeComponentINeed, ...}  
  }
```

```
</script>
```

```
<style [scoped] lang="scss | stylus | less | default = plain css">  
  ...  
</style>
```

A sneak peak!

- Our initial setup is now complete.
- Let's explore what we will ultimately build together!
 - Demo

But where do we start?

- That is a lot of functionality!
- Where do we start?
 - You get one 'gift':
 - Slides up to this point (for reference)
 - 'Starter' App.vue component file
- <https://app.box.com/v/vue-workshop-cp-1>

Starting point app

- Let's look at the 'starter' App.vue file
 - Demo

Establish Baseline: Exercise

- Set up base app for workshop
 - Get starter files:
 - Replace App.vue using supplied file
 - Remove HelloWorld.vue
 - Run the app and ensure it looks like ...

Task List

Task 1

Task 2

Task 3

Exercise: Initial Breakdown

- Break up monolithic Vue component into:
 - Main App component that will load and use three, top-level components:
 - Component for the NavBar
 - Component for Tasks
 - Component for Footer

Exercise: Further Breakdown

- Break up Tasks Vue component into:
 - Tasks component that will load and use the following components:
 - Component for the TaskListHeader
 - Component for TaskList

Exercise: Final Breakdown

- Break up TaskList Vue component into:
 - TaskList component that will load and use the TaskItem component for each Task
- Hints:
 - Create hard-coded array of strings representing task names stored in some component's state data.
 - Use v-for to iterate over that collection passing each string in the array as a prop to its own TaskItem component.

Checkpoint!

- You can download a copy of the current state of our app from here:
 - <https://app.box.com/v/vue-workshop-check-2>

Exercise: Extract Model

- Create a Task model class to represent tasks.
- Instances of Task will contain the following properties:
 - id: number (default: null)
 - name: string (default: ``)
 - private: boolean (default: true)
 - complete: boolean (default: false)
 - createdAt: Date (default: new Date())
- Refactor your app to use your new model class.
 - For your 3 existing (new) Tasks, ensure you give them unique identifiers (0, 1 and 2, respectively).
 - Hard-coded for now is fine.

Let's add a task filter!

Task List

Task 1

Task 2

Task 3

Exercise: Name Filter

- Add a TaskFilter component that looks like this:



The image shows a UI component for filtering tasks. It features a text input field with the placeholder text "Type here to filter Tasks" and a green "Clear" button to its right.

- Filter appears below header and above list.
- Typing in the entry field will filter the tasks so only those containing the entered text will be displayed.
- Clicking the clear button will clear the contents of the filter box and restore the entire list.
- Hints: New 'starter' file:
 - <https://app.box.com/v/vue-workshop-task-filter-cmp>

Let's make new Tasks!

Task List

Task 1

Task 2

Task 3

Type the name for a new task and press <Enter> to create it.

Task List

Task 1

Task 2

Task 3

My new task I am about to create...|

Exercise: New Tasks

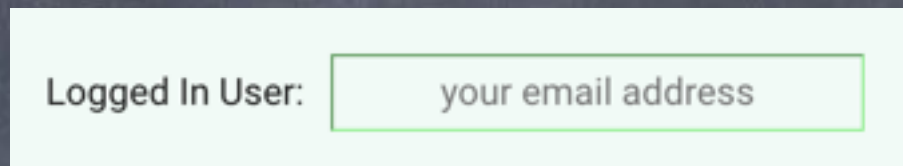
- Add a NewTask component that looks like this:

Type the name for a new task and press <Enter> to create it.

- New component appears below the task list.
- Initially shows "Type the name for a new task and press <Enter> to create it" for a placeholder.
- When the user types in the field and presses enter, create a new task (with sequential Id), and add it to the list by updating the state data. Also, set the initial collection of tasks to an empty array
- After creating the task, clear the contents and set focus back to the new task entry field, ready for the user to enter the next one.
- Hints: New 'starter' file:
 - <https://app.box.com/v/vue-workshop-new-task-cmp>

Exercise: Logging In

- Use the existing log in field that looks like this:



Logged In User:

- When the user enters their username (email or anything else) into this field and hits <Enter>, store their username in some component's state.
- When the user clears the field and hits <Enter>, log the user out by setting that state data to null.
- Use that 'logged in status' to decide whether to show the NewTask component or not, only allowing logged in users to create new Tasks.

Exercise: Owning It!

- When a logged in user creates a Task, ensure they get credit for it!
- Add an 'owner' property to the Task model and populate it with the username of the logged in user who creates it.
- Display the owner's username in parenthesis following the name of the Task in the task list

Exercise: Losing It!

- Allow task owners to delete their tasks by clicking an 'x' displayed to the far-right of the owner's username in each task list entry.
- Only display the 'x' for tasks for which the logged in user is the owner.

Problem: Passing the Buck

The shell game

- We have to pass 'username' around to virtually EVERY component in our system.
- Way to much chance for error and far too much boilerplate code to write.

Playing 'telephone'

- We have to manually propagate the 'delete' event all the way up from the inner-most component to the top-level component.
- Too much boilerplate and opportunity to err.

Solution:
Vuex

What is Vuex?

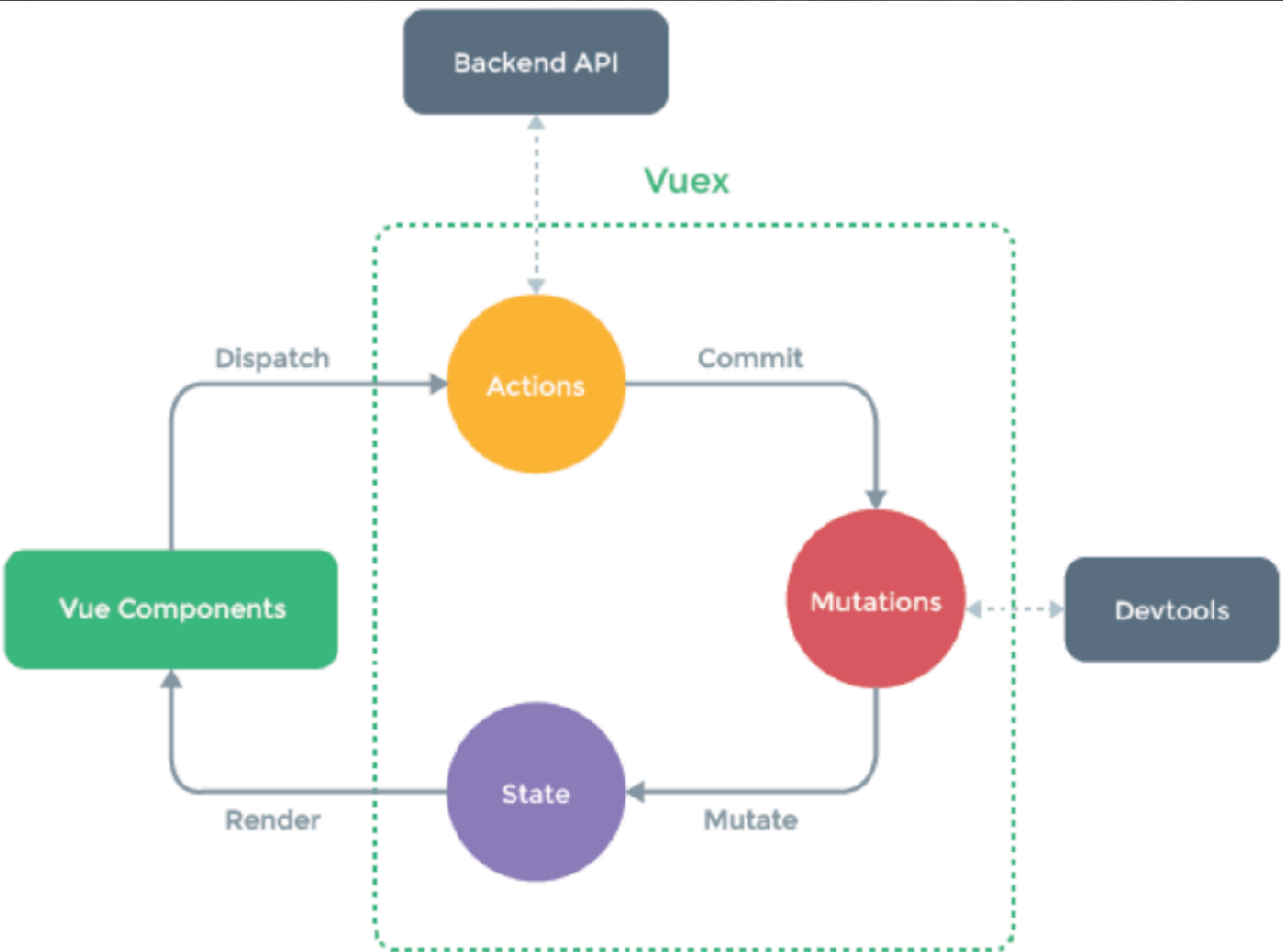
- A central 'store' for your application state.
- Data is 'safe'
 - No changes without sending notifications
 - Very predictable change management
- Easy access
 - Convenient accessing from any component

What is Vuex?

- Implements a 'one-way' data update flow
 - State is held in 'trust'
 - Views render a copy of that state.
 - Actions are 'dispatched'
 - From user interaction or external events.
 - Can result in API calls being made

What is Vuex?

- Implements a 'one-way' data flow (continued)
 - The 'store' reacts to those actions
 - Updates the data in the 'store'
 - Emits 'events' documenting the changes
- The views are (partially) re-rendered based on the changes signaled by the events emitted by the store.



When should you use it?

- Quote from Dan Abramov (author of Redux):

“Flux libraries are like glasses:
you’ll know when you need them”

Using Vuex

- Install Vuex
 - `vue add vuex`
- Configure vuex
 - Create a 'store' file
 - Similar to creating a Vue component
 - Import the 'store'
 - Tell Vue to use it

Installing Vuex

- Vuex is an official plug-in for Vue

```
➔ taskmaster git:(master) vue add vuex
```

```
🚀 Invoking generator for core:vuex...
```

```
📦 Installing additional dependencies...
```

```
yarn install v1.7.0
```

```
[1/4] 🔍 Resolving packages...
```

```
[2/4] 🚚 Fetching packages...
```

```
[3/4] 🔗 Linking dependencies...
```

```
[4/4] 📦 Building fresh packages...
```

```
success Saved lockfile.
```

```
✨ Done in 3.25s.
```

```
Successfully invoked generator for plugin: core:vuex
```

```
hasProjectGit /Users/pavlovich/Documents/Presentations/vue/workshop/src/taskmaster true
```

```
The following files have been updated / added:
```

```
src/store.js
```

```
package.json
```

```
src/main.js
```

```
yarn.lock
```

```
You should review these changes with git diff and commit them.
```

```
➔ taskmaster git:(master) x █
```

Configure Vuex: store.js

```
import Vue from 'vue'  
import Vuex from 'vuex'
```

```
Vue.use(Vuex)
```

```
export default new Vuex.Store({  
  state: { // holds the actual state in 'Trust'  
    counter: 23, todos: [new Todo({...}), new Todo({...}), ...]  
  },  
  mutations: { // updates the store directly. SYNCHRONOUS ONLY  
    changeCounter(state, amount){state.counter = counter + amount}  
  },  
  getters: { // like computed props on a component  
    doneTodos: (state, args) => {  
      return state.todos.filter(todo => todo.done)  
    }  
  }  
  actions: { // Can do async calls. Calls mutations via #commit  
    increaseCounter: (context, amount) => {  
      return context.commit('changeCounter', amount)  
    }  
  }  
})
```


Accessing the store

- inside a component
 - `this.$store`
- Outside a component
 - `import store from `.../store`;`

Dispatching Actions

- Get hold of the store and call 'dispatch'

```
actions: {  
  increment ({ commit }) {  
    commit('increment')  
  }  
}
```

js

```
store.dispatch('increment')
```

js

Dispatching Actions

```
actions: {  
  incrementAsync ({ commit }) {  
    setTimeout(() => {  
      commit('increment')  
    }, 1000)  
  }  
}
```

js

```
// dispatch with a payload  
store.dispatch('incrementAsync', {  
  amount: 10  
})  
  
// dispatch with an object  
store.dispatch({  
  type: 'incrementAsync',  
  amount: 10  
})
```

js

Access state in components

```
import { mapState } from 'vuex'

export default {
  // ...
  computed: {
    count () {
      return this.$store.state.count
    }
  }
}
```

```
export default {
  // ...
  computed: mapState({
    count: state => state.count,
    countAlias: 'count',

    // Use a normal function to access local state with 'this'
    countPlusLocalState (state) {
      return state.count + this.localCount
    }
  })
}
```

Access state in components

```
import { mapState } from 'vuex'

export default {
  // ...
  computed: {
    localComputed () { /* ... */ },
    // mix into the outer object with the object spread operator
    ...mapState({
      count: state => state.count,
      countAlias: 'count',

      // Use a normal function to access local state with 'this'
      countPlusLocalState (state) {
        return state.count + this.localCount
      }
    })
  }
}
```

Access state in components

```
import { mapState } from 'vuex'

export default {
  // ...
  computed: {
    localComputed () { /* ... */ },
    // map this.counter to store.state.counter
    // map this.tasks to store.state.tasks
    ...mapState(['counter', 'tasks'])
  }
}
```


Defining getters

```
const store = new Vuex.Store({
  state: {
    todos: [
      { id: 1, text: '...', done: true },
      { id: 2, text: '...', done: false }
    ]
  },
  getters: {
    doneTodos: state => {
      return state.todos.filter(todo => todo.done)
    },
    doneTodosCount: (state, getters) => {
      return getters.doneTodos.length
    },
    getTodoById: (state) => (id) => {
      return state.todos.find(todo => todo.id === id)
    }
  }
})
```

Accessing getters

```
import { mapGetters } from 'vuex'

export default {
  // ...
  computed: {
    // mix the getters into computed with object spread operator
    ...mapGetters([
      'doneTodosCount',
      'anotherGetter',
      // ...
    ])
  }
}

OR

computed: {
  ...mapGetters({
    // map `this.doneCount` to `this.$store.getters.doneTodosCount`
    doneCount: 'doneTodosCount'
  })
}
}
```

Accessing mutations

```
import { mapMutations } from 'vuex'

export default {
  // ...
  methods: {
    ...mapMutations([
      'increment',
      // map `this.increment()` to
      // `this.$store.commit('increment')`

      // `mapMutations` also supports payloads:
      'incrementBy'
      // map `this.incrementBy(amount)` to
      // `this.$store.commit('incrementBy', amount)`
    ]),
    ...mapMutations({
      add: 'increment'
      // map `this.add()` to
      // `this.$store.commit('increment')`
    })
  }
}
```


Accessing Actions

```
import { mapActions } from 'vuex'

export default {
  // ...
  methods: {
    ...mapActions([
      'increment',
      // map `this.increment()` to
      // `this.$store.dispatch('increment')`

      // `mapActions` also supports payloads:
      'incrementBy'
      // map `this.incrementBy(amount)` to
      // `this.$store.dispatch('incrementBy', amount)`
    ]),
    ...mapActions({
      add: 'increment'
      // map `this.add()` to
      // `this.$store.dispatch('increment')`
    })
  }
}
```

Handling 2-way binding

```
<input v-model="message">
```

html

```
// ...  
computed: {  
  message: {  
    get () {  
      return this.$store.state.obj.message  
    },  
    set (value) {  
      this.$store.commit('updateMessage', value)  
    }  
  }  
}
```

js

Other considerations

- Prefer initializing your store's initial state with all desired fields upfront.
- When adding new properties to an Object in the store, you should either:
 - Use `Vue.set(obj, 'newProp', 123)`, or
 - Replace that Object with a fresh one as in:

```
state.obj = { ...state.obj, newProp: 123 }
```

js

Exercise: Store It!

- Add Vuex to your project
- Create a store in a new store file (root of proj)
- Move 'username', 'nextId', 'tasks', 'newTask', 'filterString' into Vuex store.
- Hints:
 - Download slides + state of app at this point
 - <https://app.box.com/v/vue-workshop-third-check>

Routing

Vue's core router

- 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

Installation

- Another core add-on:
 - `vue add router`

Installation

```
→ taskmaster git:(master) vue add router
```

```
🚀 Invoking generator for core:router...
```

```
📦 Installing additional dependencies...
```

```
yarn install v1.7.0
```

```
[1/4] 🔍 Resolving packages...
```

```
[2/4] 📦 Fetching packages...
```

```
[3/4] 🔗 Linking dependencies...
```

```
[4/4] 🏗 Building fresh packages...
```

```
success Saved lockfile.
```

```
🌟 Done in 2.87s.
```

```
Successfully invoked generator for plugin: core:router
```

```
hasProjectGit /Users/pavlovich/Documents/Presentations/vue/workshop/src/taskmaster true
```

```
The following files have been updated / added:
```

```
src/router.js
src/views/About.vue
src/views/Home.vue
package.json
src/App.vue
src/main.js
yarn.lock
```

```
You should review these changes with git diff and commit them.
```

```
→ taskmaster git:(master) x █
```

Group Exercise: Add Routing

- Demo

WRAP UP!!!

Thank You!

www.vuejs.org

pavlovich@gmail.com

<http://linkedin.com/in/peterpavlovich>

<https://github.com/pavlovich/vue-workshop>

Resources

- Main site: <http://www.vuejs.org>
- Add-ons:
 - Vuetify: <https://vuetifyjs.com>
 - Vuex: <https://vuex.vuejs.org>
 - Router: <https://router.vuejs.org>
 - Nuxt: <https://nuxtjs.org>
- Curated Lists:
 - <https://curated.vuejs.org>
 - <https://github.com/nuxt-community/awesome-nuxt>
- New feeds:
 - <https://www.vuejsradar.com>
 - <https://github.com/vuejs/awesome-vue>

Courses

- Udemy:
 - Anything by Max!
 - <https://www.udemy.com/vuejs-2-the-complete-guide>
 - <https://www.udemy.com/nuxtjs-vuejs-on-steroids>

Max's courses are awesome! Some graphics came from his courses.

Highly recommended!

Questions?