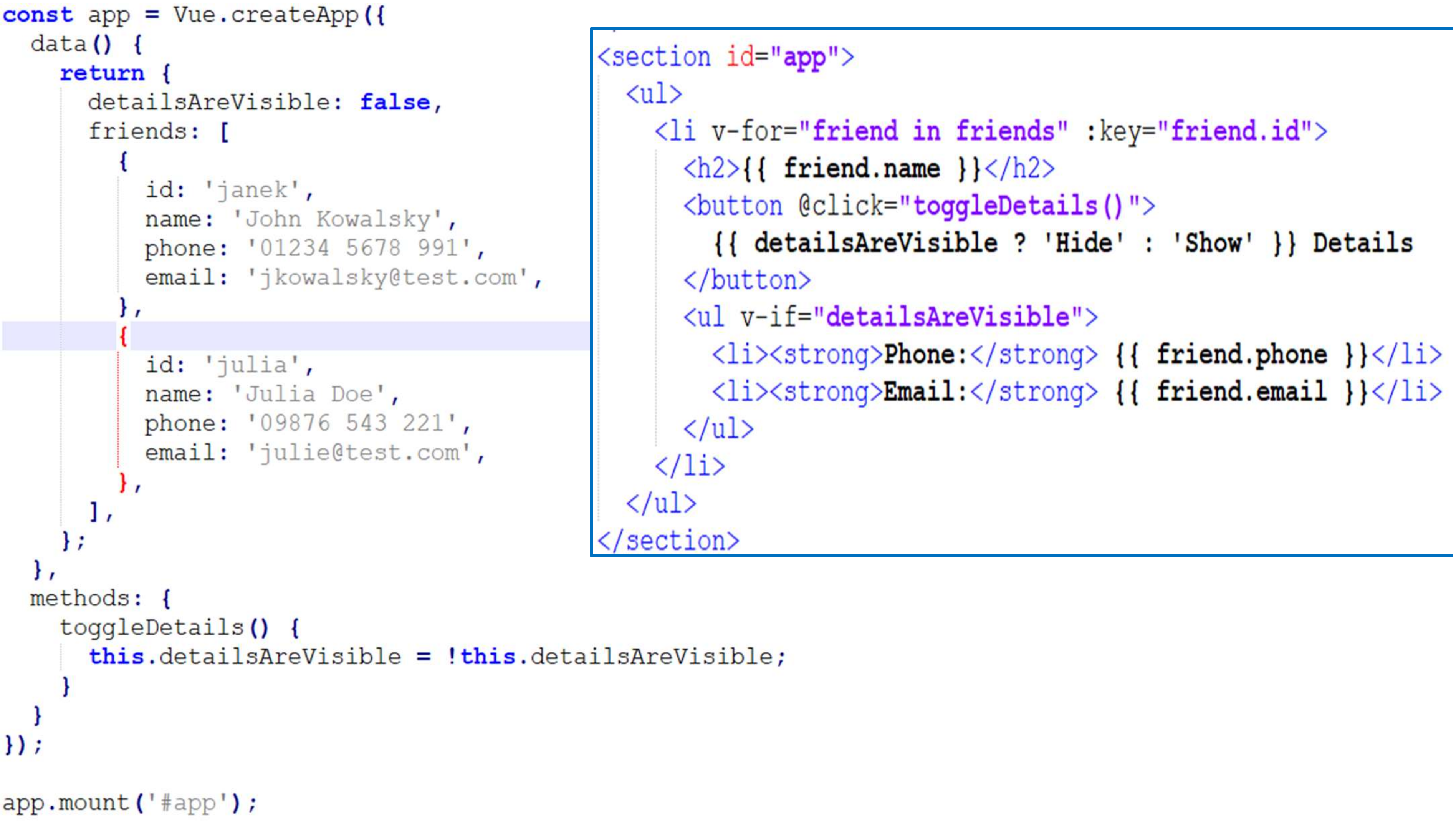
Advanced Web Programming

Component in Vue 3

Grzegorz Rogus
rogus@agh.edu.pl

Akademia Górniczo-Hutnicza im. Stanisława Staszica w Krakowie
AGH University of Science and Technology

Problems with Vue Instance

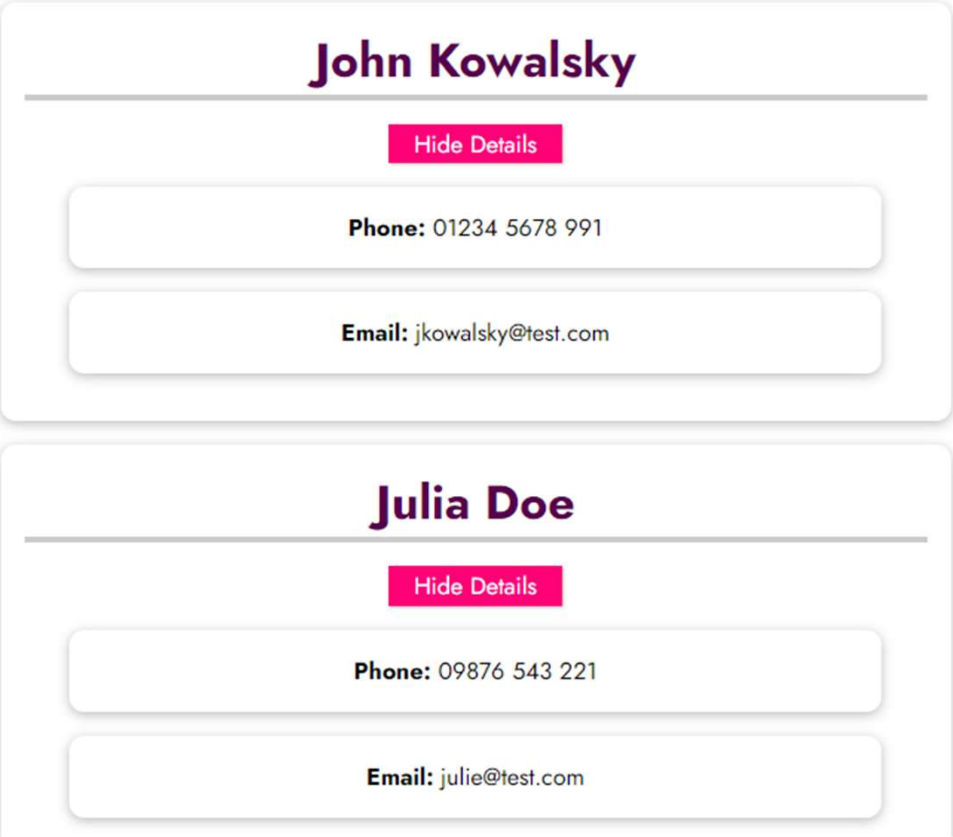


```
const app = Vue.createApp({
  data() {
    return {
      detailsAreVisible: false,
      friends: [
        {
          id: 'janek',
          name: 'John Kowalsky',
          phone: '01234 5678 991',
          email: 'jkowalsky@test.com',
        },
        {
          id: 'julia',
          name: 'Julia Doe',
          phone: '09876 543 221',
          email: 'julie@test.com',
        },
      ],
    };
  },
  methods: {
    toggleDetails() {
      this.detailsAreVisible = !this.detailsAreVisible;
    }
  }
});

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

```
<section id="app">
  <ul>
    <li v-for="friend in friends" :key="friend.id">
      <h2>{{ friend.name }}</h2>
      <button @click="toggleDetails()">
        {{ detailsAreVisible ? 'Hide' : 'Show' }} Details
      </button>
      <ul v-if="detailsAreVisible">
        <li><strong>Phone:</strong> {{ friend.phone }}</li>
        <li><strong>Email:</strong> {{ friend.email }}</li>
      </ul>
    </li>
  </ul>
</section>
```

Problems with Vue Instance

Demo -> Example1



John Kowalsky

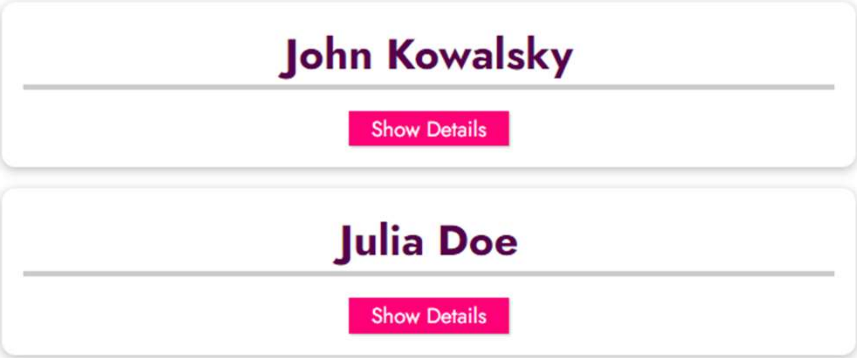
Phone: 01234 5678 991

Email: jkowalsky@test.com

Julia Doe

Phone: 09876 543 221

Email: julie@test.com



John Kowalsky

Julia Doe

Multiple Vue Apps vs Multiple Components

If you control multiple, independent parts of HTML pages, you will work with multiple Vue apps (use Vue.js to control **parts** of pages)

You typically won't use multiple Vue apps if you build one big connected user interface (or "**Single Page Applications**" (**SPA**s).)

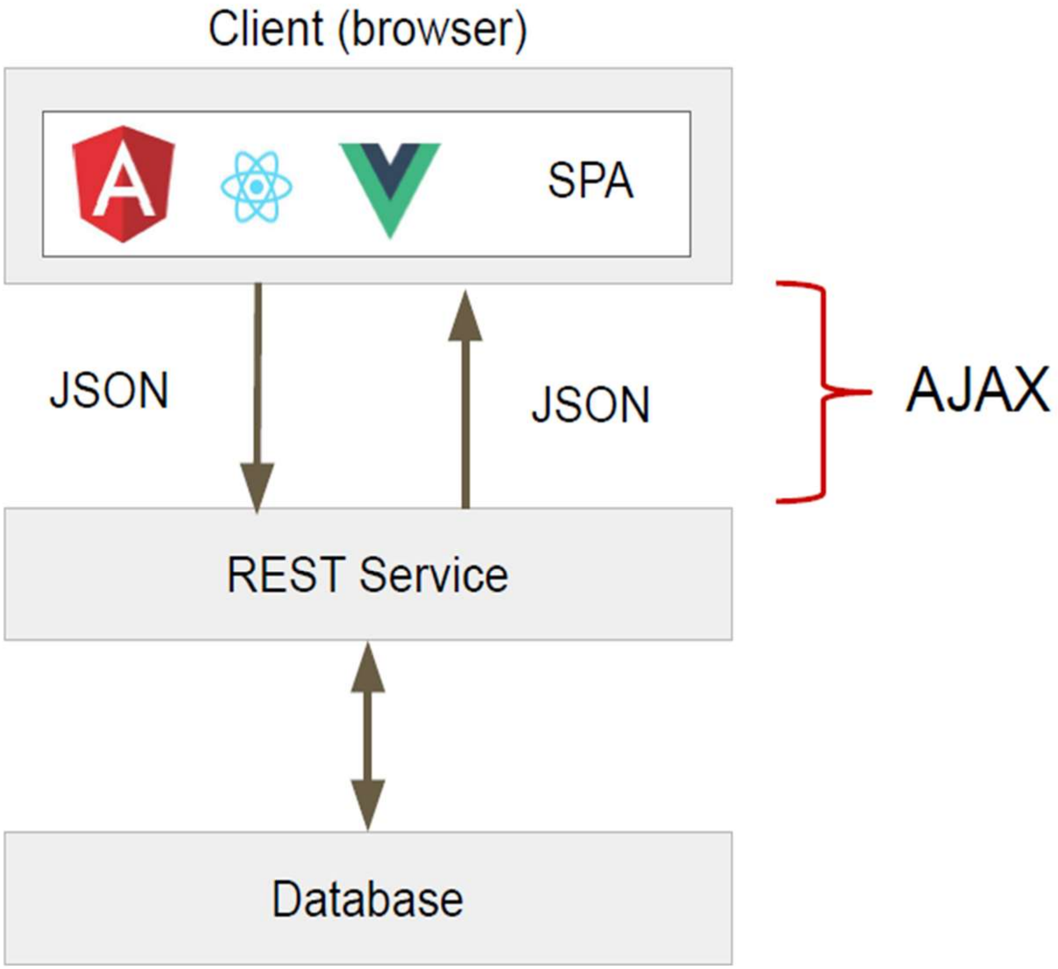
Why??

Because Vue apps are independent from each other - they can't really communicate with each other.

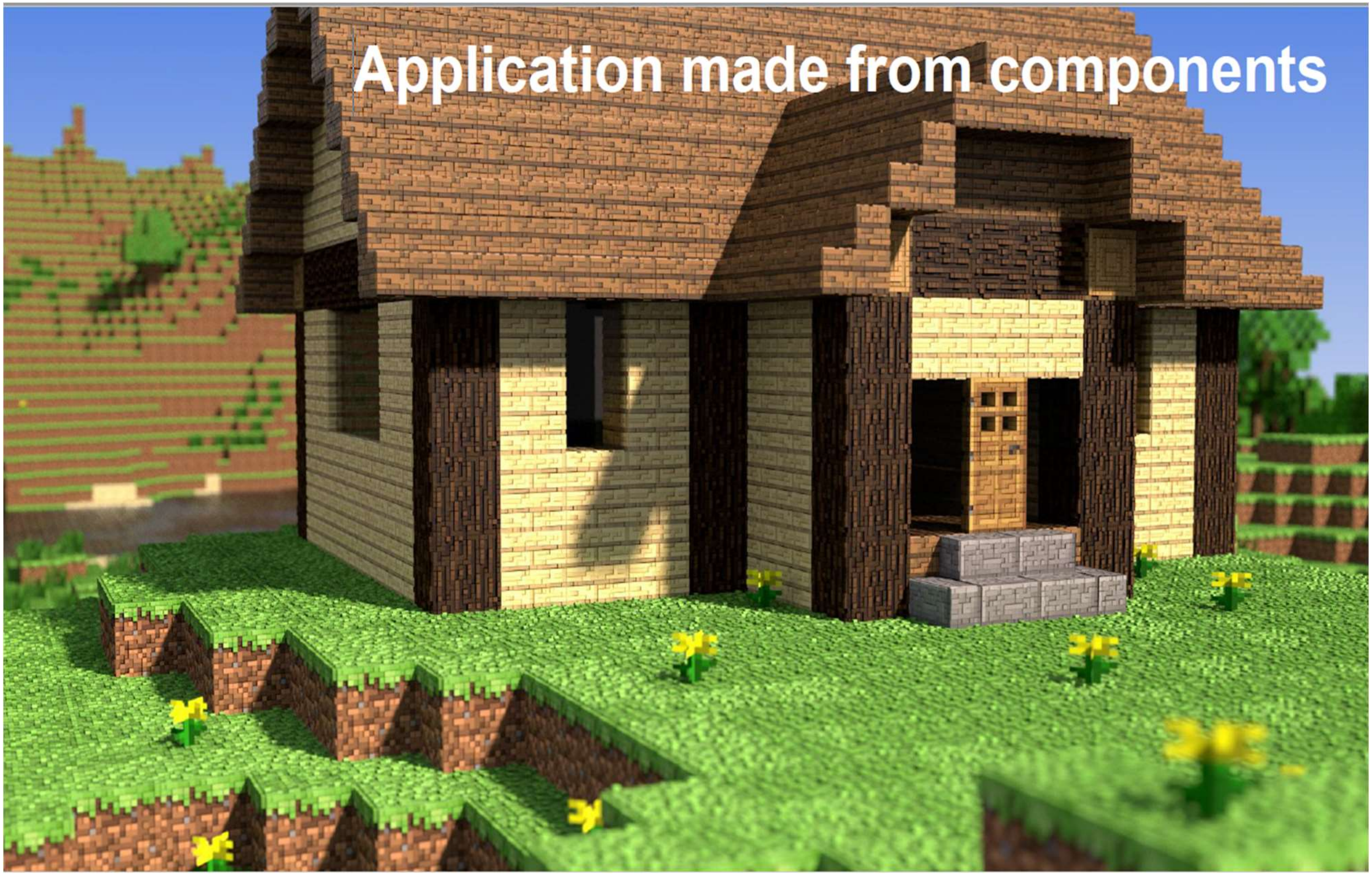
So if you're building a SPA, you have to work with just one "root app" (and you instead build up a user interface with multiple components.

Components on the other hand - as you will learn soon - DO offer certain communication mechanisms that allow you to exchange data between them.

Single Page Application (SPA)



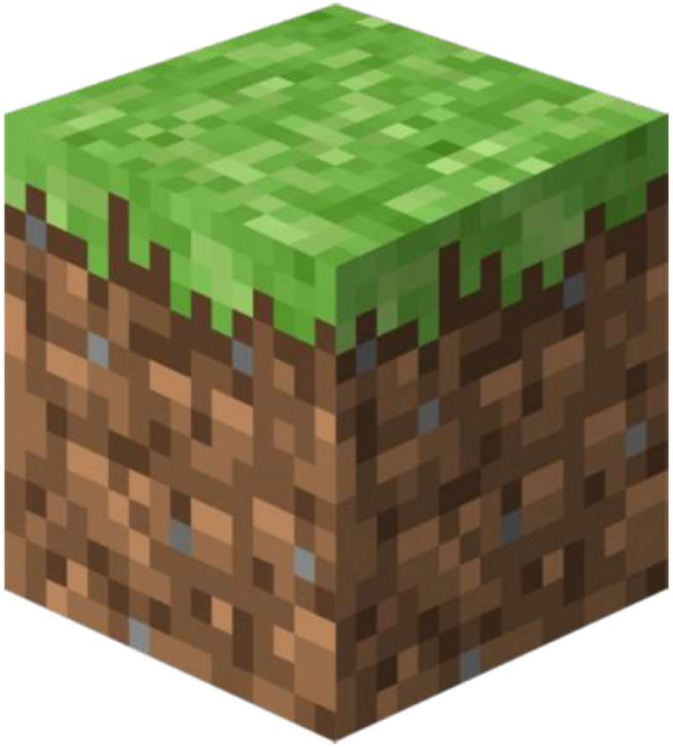
Application made from components



Components are single, independent units of an interface.
They can have their own state, markup and style.

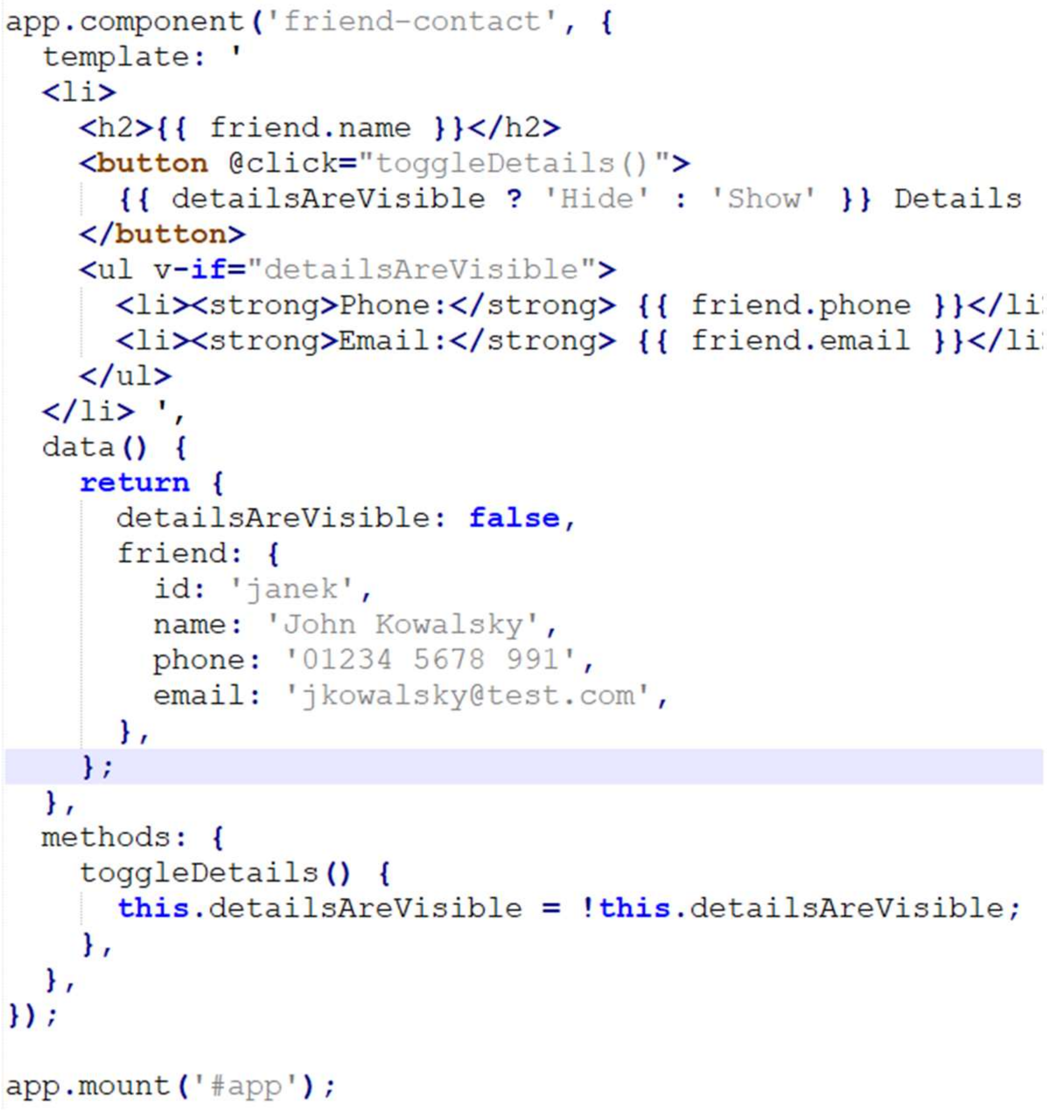
Component

- have structure (HTML)
- behave and have a state (JS)
- appeal (CSS)
- have single functionality (SRP rule – Single Responsibility Principle)

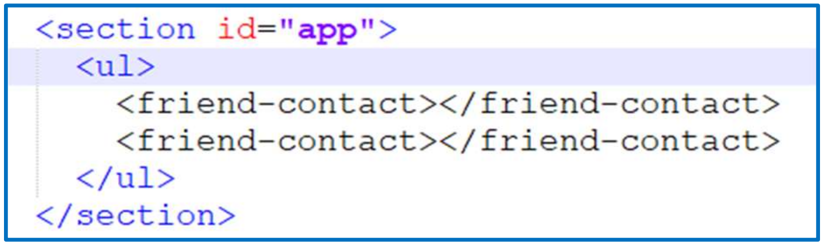


Solution – Components?

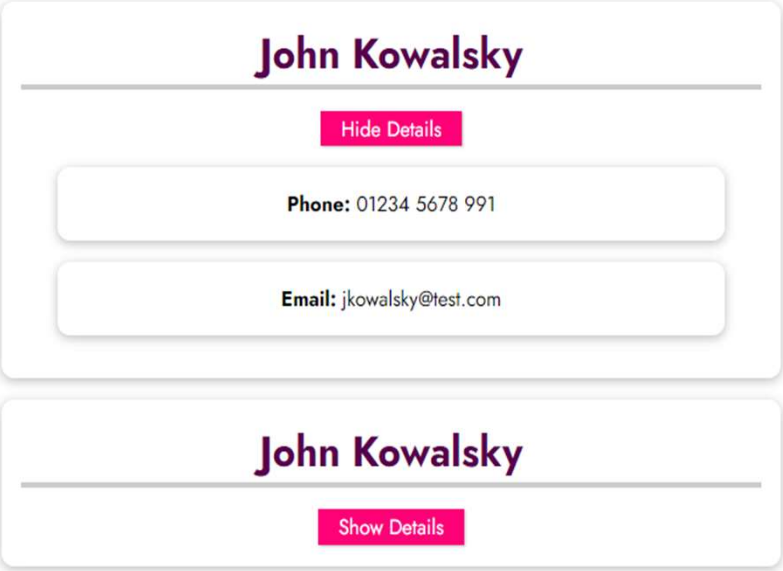
// Create a Vue application
const app = Vue.createApp({})
// Define a new global component



```
app.component('friend-contact', {
  template: '
  <li>
    <h2>{{ friend.name }}</h2>
    <button @click="toggleDetails()">
      {{ detailsAreVisible ? 'Hide' : 'Show' }} Details
    </button>
    <ul v-if="detailsAreVisible">
      <li><strong>Phone:</strong> {{ friend.phone }}</li
      <li><strong>Email:</strong> {{ friend.email }}</li
    </ul>
  </li> ',
  data() {
    return {
      detailsAreVisible: false,
      friend: {
        id: 'janek',
        name: 'John Kowalsky',
        phone: '01234 5678 991',
        email: 'jkowalsky@test.com',
      },
    };
  },
  methods: {
    toggleDetails() {
      this.detailsAreVisible = !this.detailsAreVisible;
    },
  },
});

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



```
<section id="app">
  <ul>
    <friend-contact></friend-contact>
    <friend-contact></friend-contact>
  </ul>
</section>
```

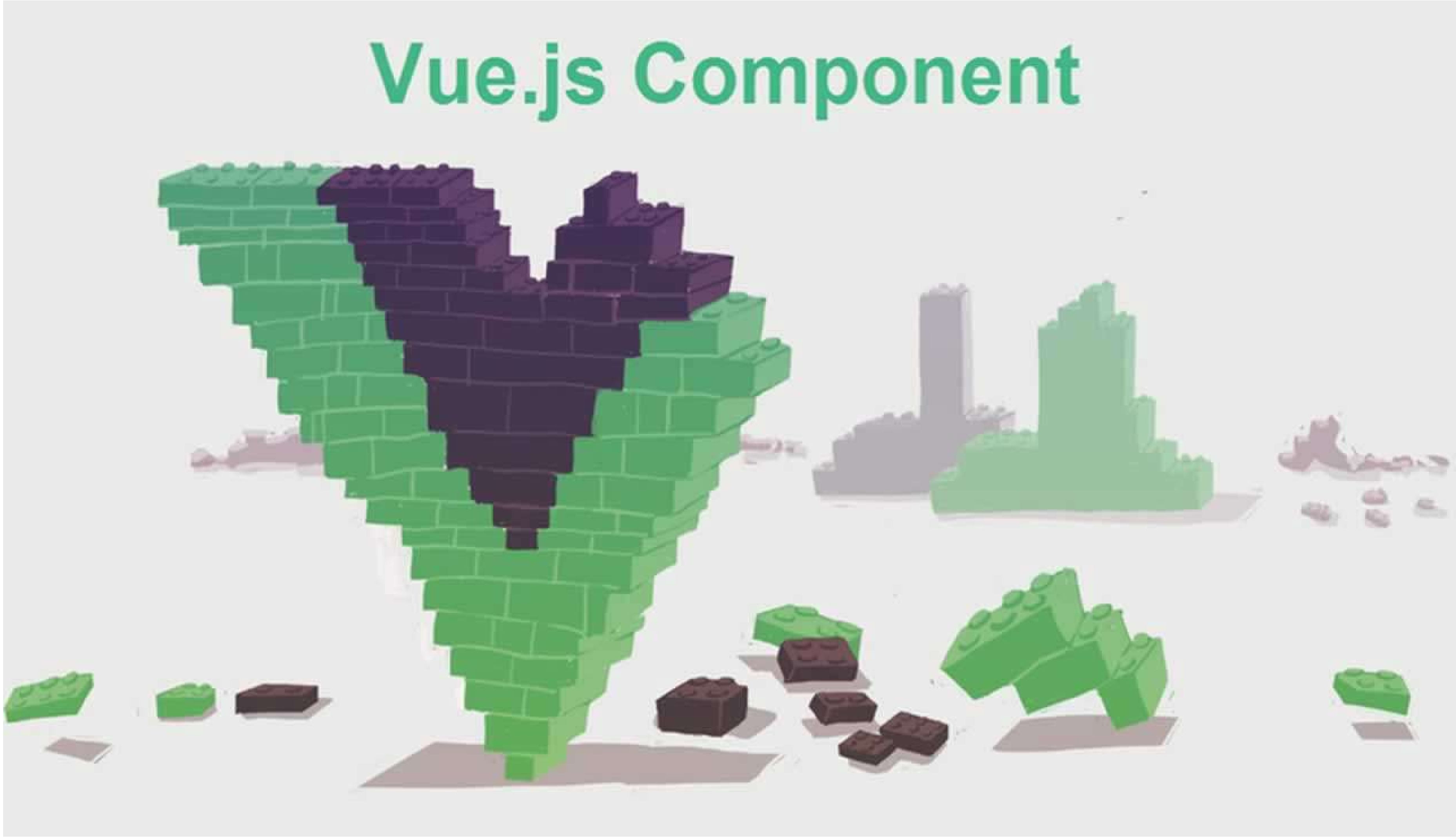


John Kowalsky

Phone: 01234 5678 991

Email: jkowalsky@test.com

John Kowalsky



The most of Vue app is built from components.

Components in Vue

- Components are a central part of building apps in Vue.
- These components let you break a large application into discrete building blocks that can be created and managed separately, and transfer data between each other as required.
- These small blocks can help you reason about and test your code.

Properties of Component

- Small
- Self-contained
- Often reusable

How to build SPA App in Vue.

- Strong recomended method to build SPA app in Vue 3 is use CLI.
- Vue provides an official CLI for quickly scaffolding ambitious Single Page Applications. It provides batteries-included build setups for a modern frontend workflow.

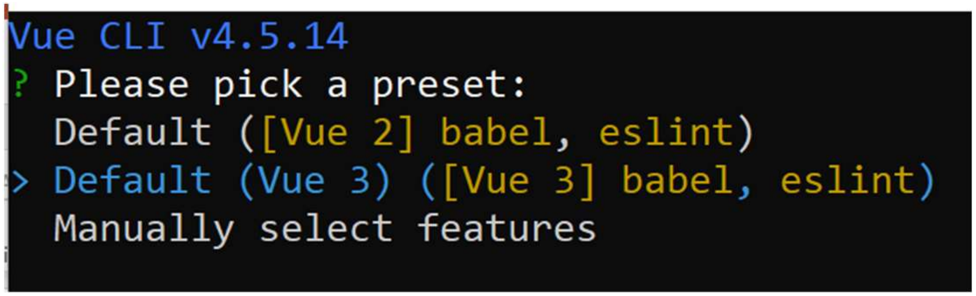
- To use the CLI you will need:

node.js installed.

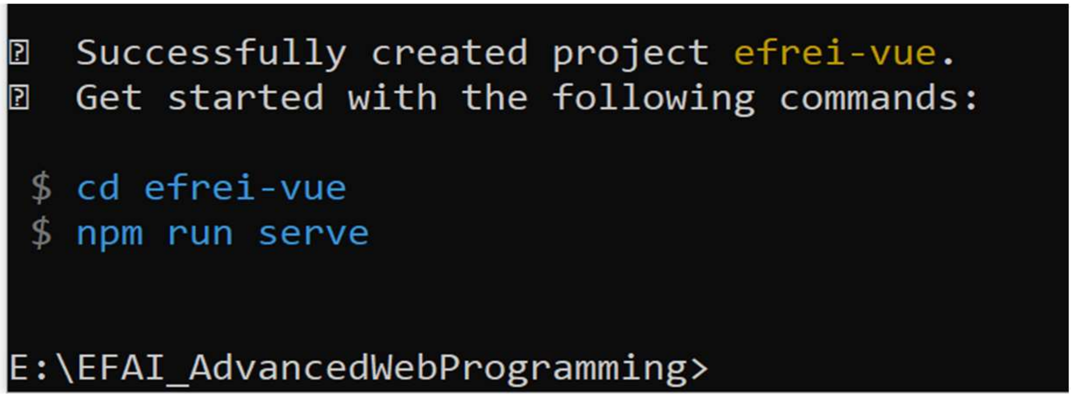
npm or yarn as a package manager for JS packages or modules,

Vue.js - initializing a new project

- install Node.js
- npm install -g @vue/cli
- vue create efrei-vue



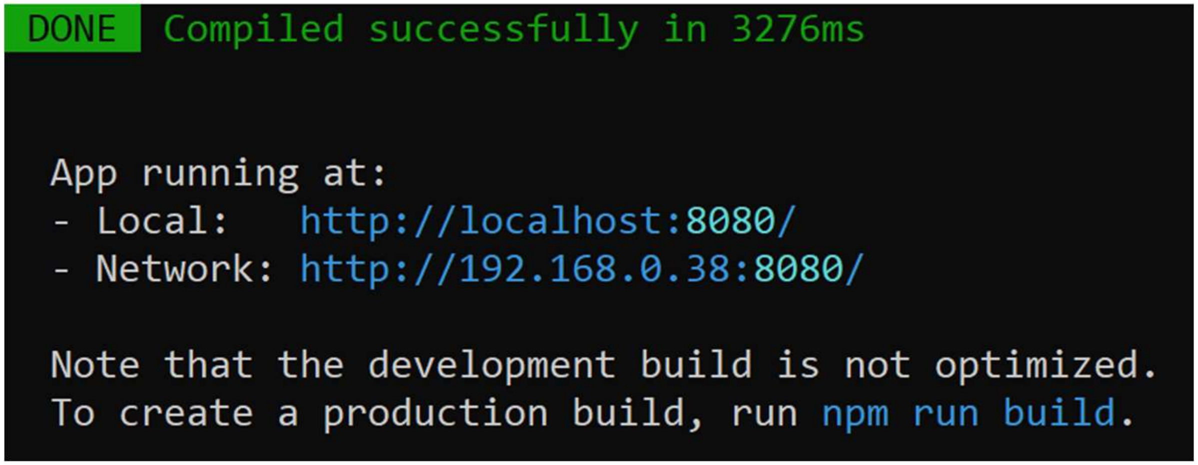
```
Vue CLI v4.5.14
? Please pick a preset:
  Default ([Vue 2] babel, eslint)
> Default (Vue 3) ([Vue 3] babel, eslint)
  Manually select features
```



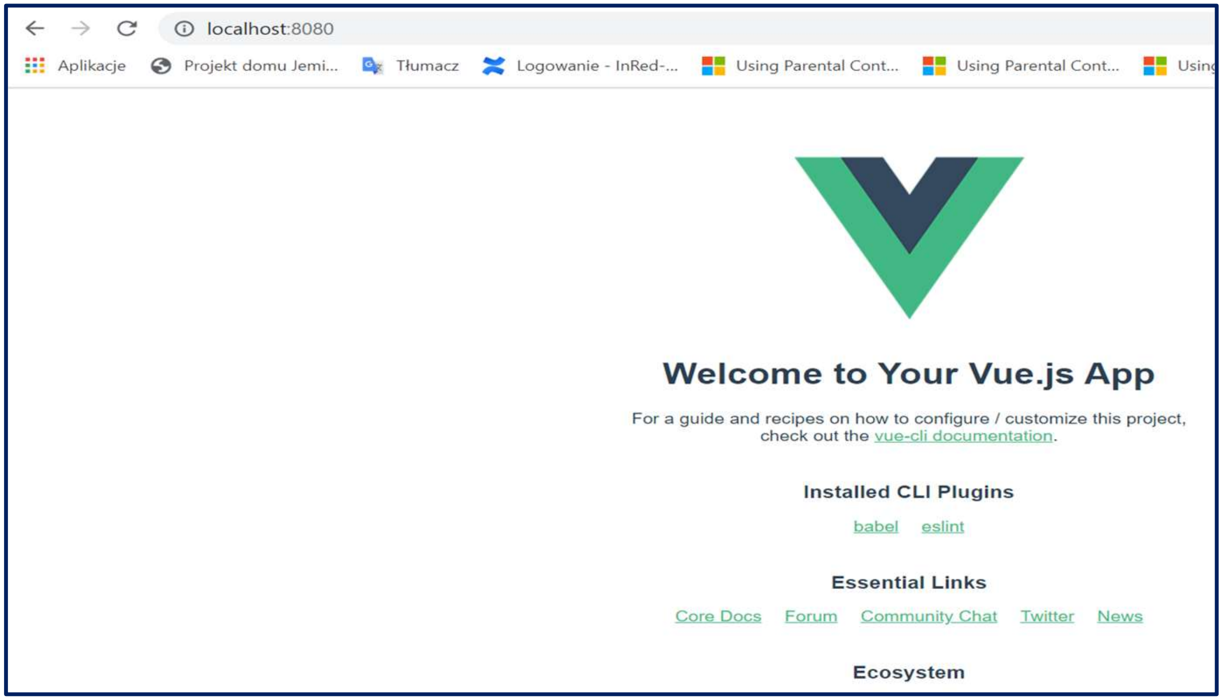
```
⍰  Successfully created project efrei-vue.
⍰  Get started with the following commands:

 $ cd efrei-vue
 $ npm run serve


E:\EFAI_AdvancedWebProgramming>
```

- cd efrei-vue
- npm run serve

Runing Vue project cd.



```
 DONE  Compiled successfully in 3276ms


  App running at:
  - Local:   http://localhost:8080/
  - Network: http://192.168.0.38:8080/

  Note that the development build is not optimized.
  To create a production build, run npm run build.
```



Welcome to Your Vue.js App

For a guide and recipes on how to configure / customize this project, check out the vue-cli documentation.

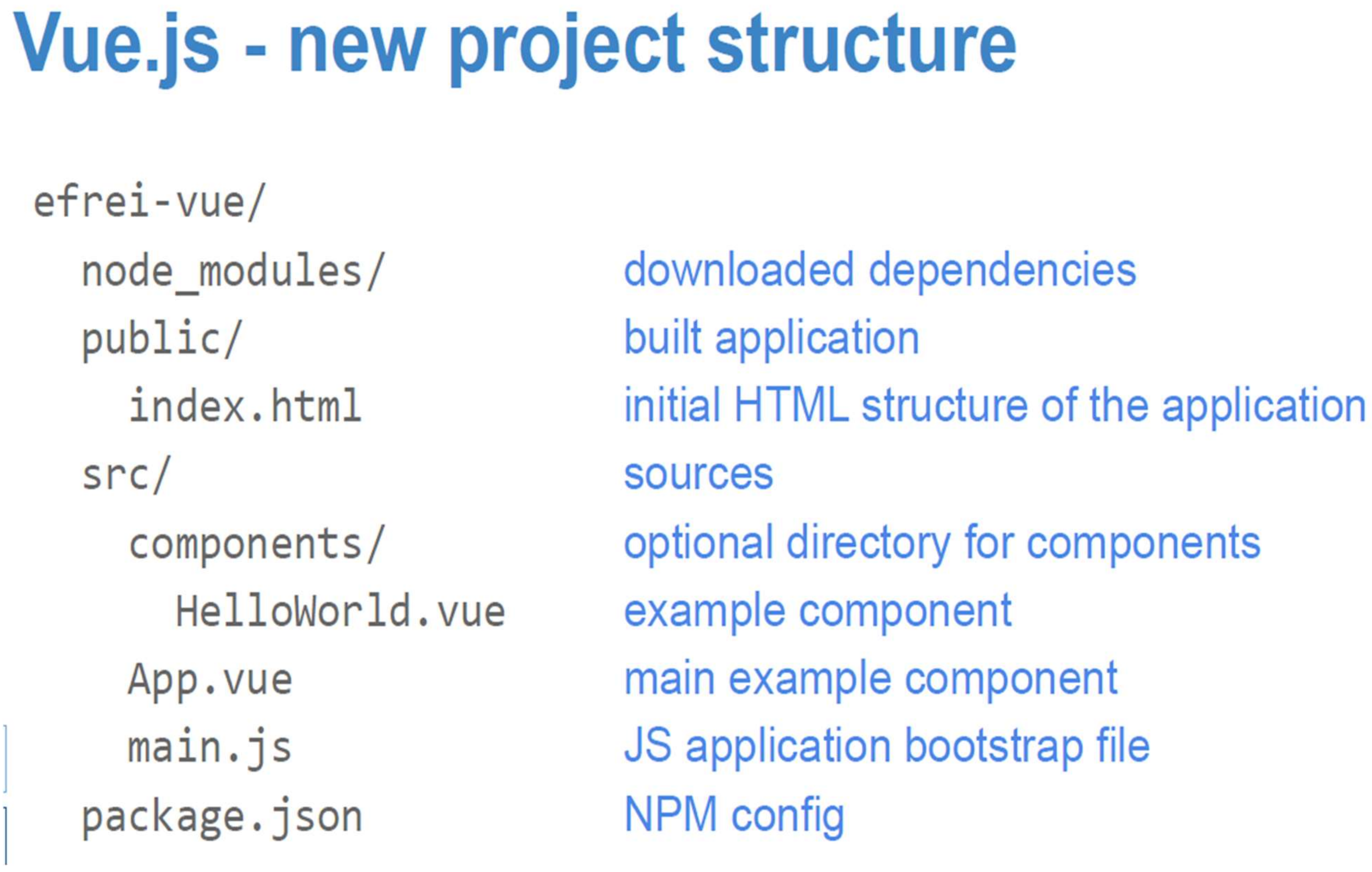
Installed CLI Plugins

babel eslint

Essential Links

Core Docs Forum Community Chat Twitter News

Ecosystem

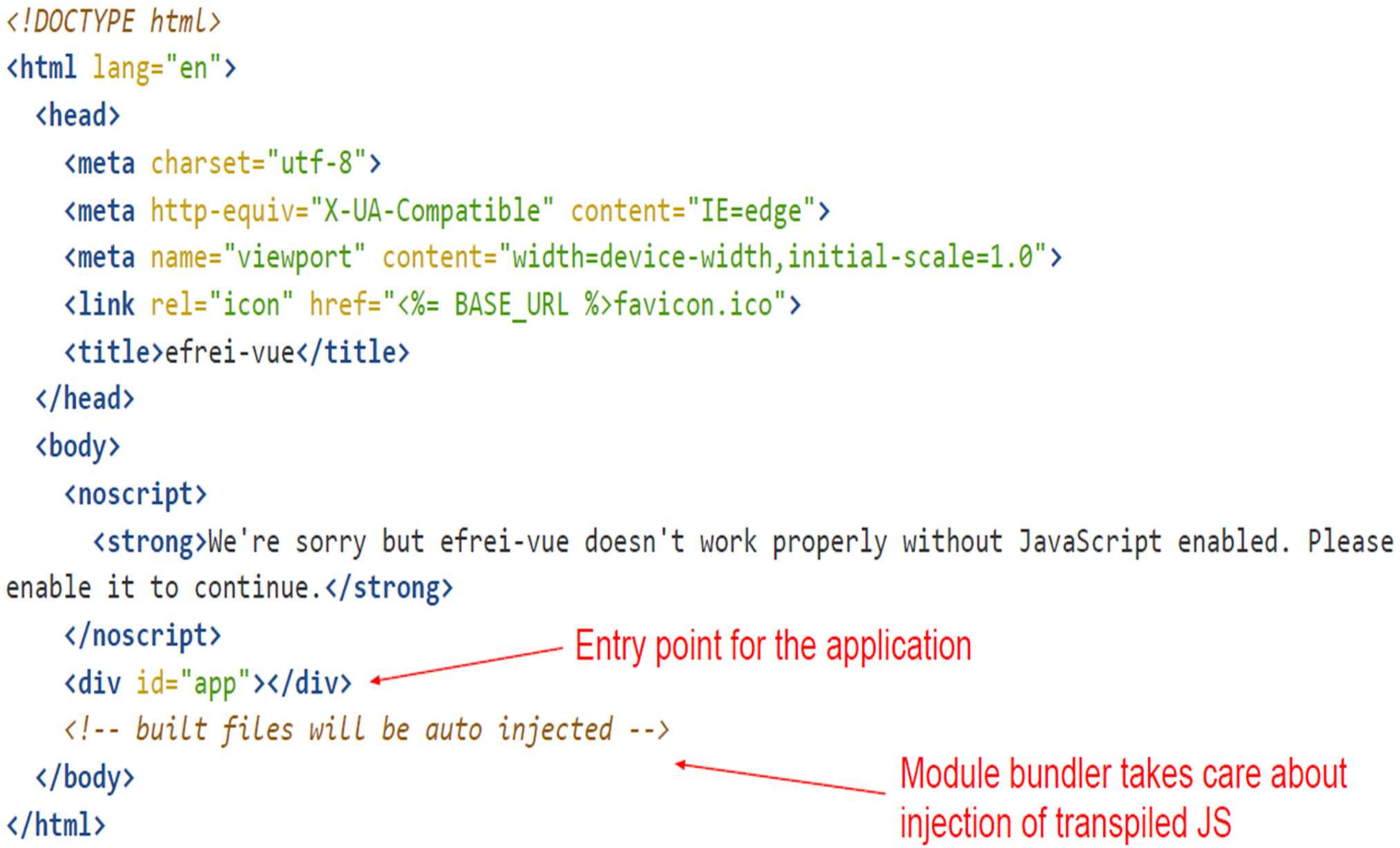


Vue.js - new project structure

```
efrei-vue/
  node_modules/           downloaded dependencies
  public/                 built application
    index.html            initial HTML structure of the application
  src/                    sources
    components/           optional directory for components
      HelloWorld.vue      example component
    App.vue               main example component
    main.js               JS application bootstrap file
  package.json            NPM config
```

How it works?

1. Start with public/index.html



```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width,initial-scale=1.0">
    <link rel="icon" href="<%= BASE_URL %>favicon.ico">
    <title>efrei-vue</title>
  </head>
  <body>
    <noscript>
      <strong>We're sorry but efrei-vue doesn't work properly without JavaScript enabled. Please
enable it to continue.</strong>
    </noscript>
    <div id="app"></div>
    <!-- built files will be auto injected -->
  </body>
</html>
```

Entry point for the application

Module bundler takes care about injection of transpiled JS

2. Main js file: src/main.js

```
import { createApp } from 'vue'
import App from './App.vue’

createApp(App).$mount('#app')
```

Starts the application with App component in
the #app element (remember index.html?)

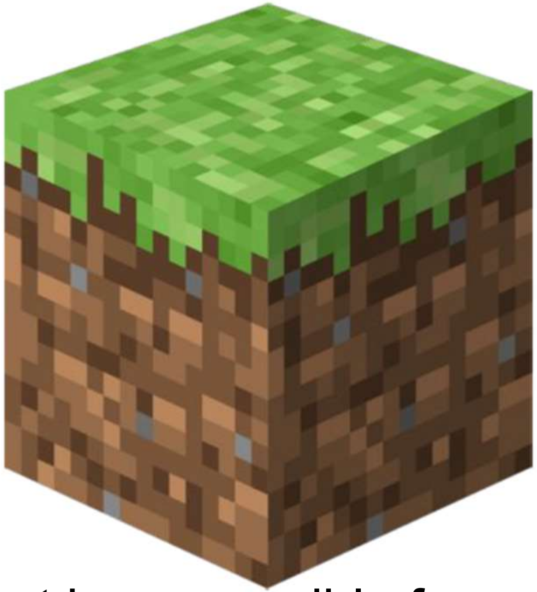
App – it is a root component

The options passed to createApp are used to configure the root component. That component is used as the starting point for rendering when we mount the application.

src/App.vue - the component

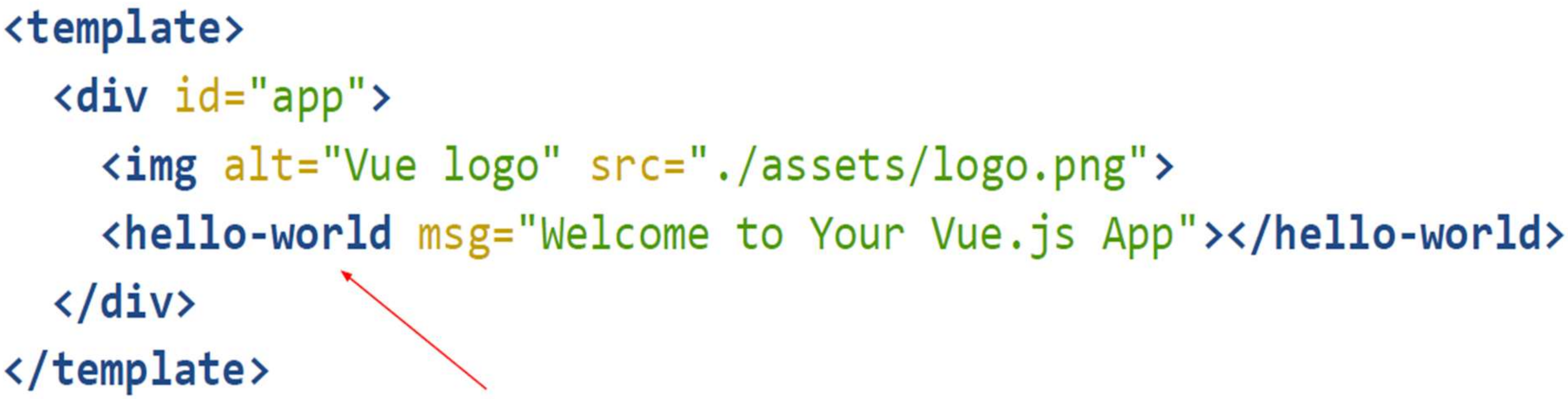
- has structure (HTML)
- behaves and has a state (JS)
- appeals (CSS)

```
<template>
  <div>
   ...
  </div>
</template>

<script>
export default {
}
</script>

<style>
p {
}
</style>
```



Single File Components that is responsible for everything that regards a single component, centralizing the responsibility for the appearance and behavior

src/App.vue - structure

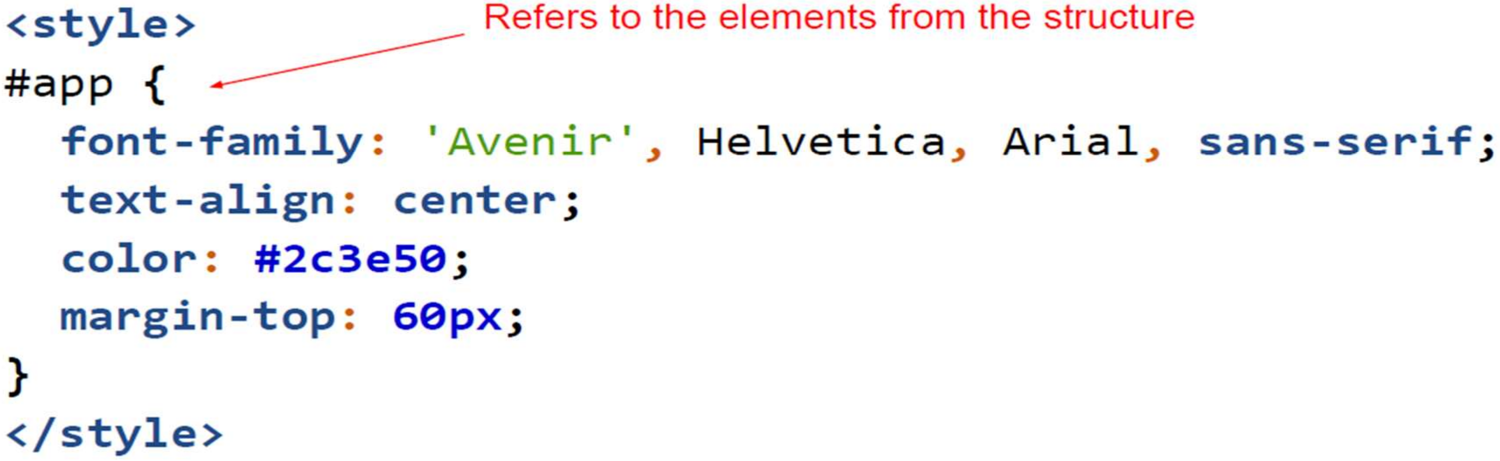


```
<template>
  <div id="app">
    <img alt="Vue logo" src="./assets/logo.png">
    <hello-world msg="Welcome to Your Vue.js App"></hello-world>
  </div>
</template>
```

Components can use other components!

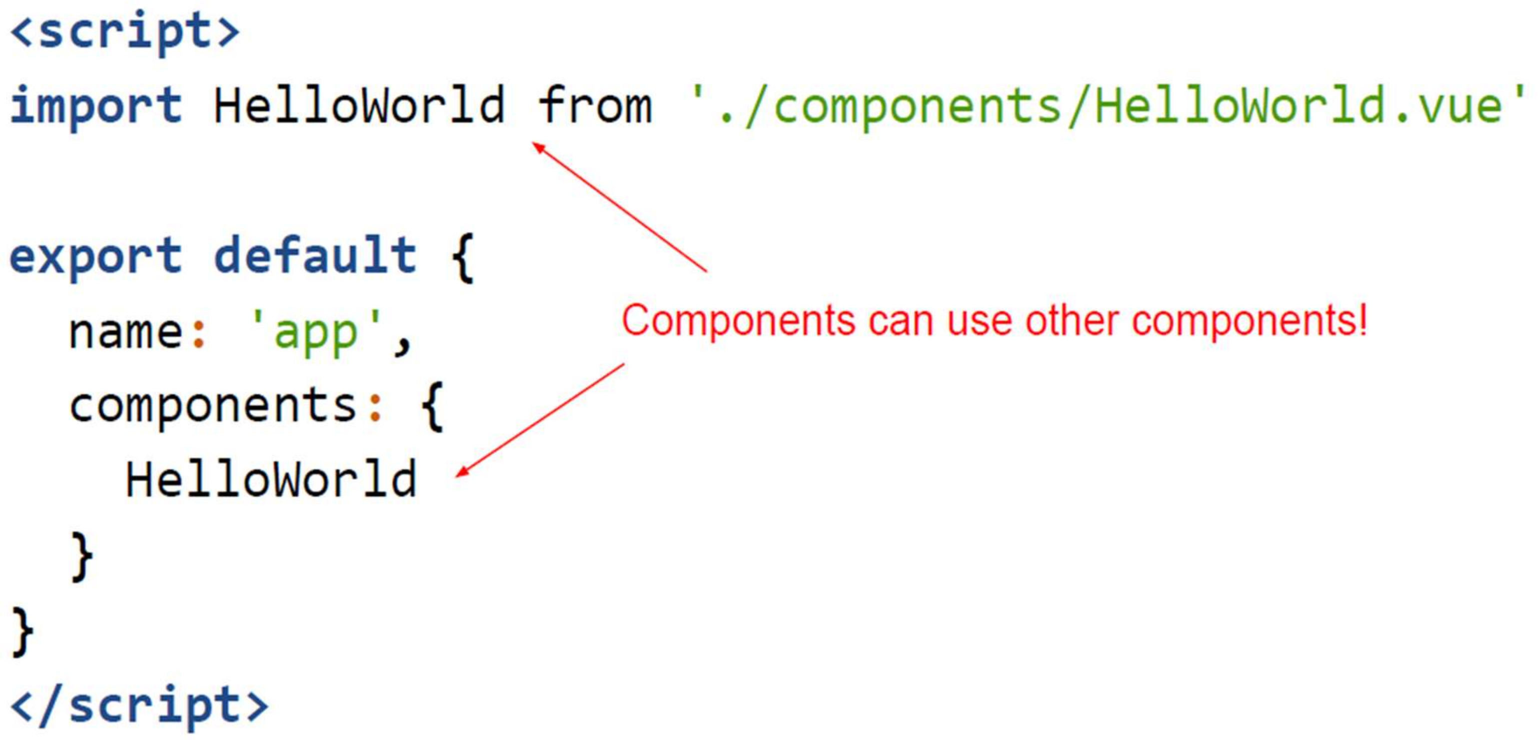
src/App.vue - appearance

Refers to the elements from the structure



```
<style>
#app {
  font-family: 'Avenir', Helvetica, Arial, sans-serif;
  text-align: center;
  color: #2c3e50;
  margin-top: 60px;
}
</style>
```

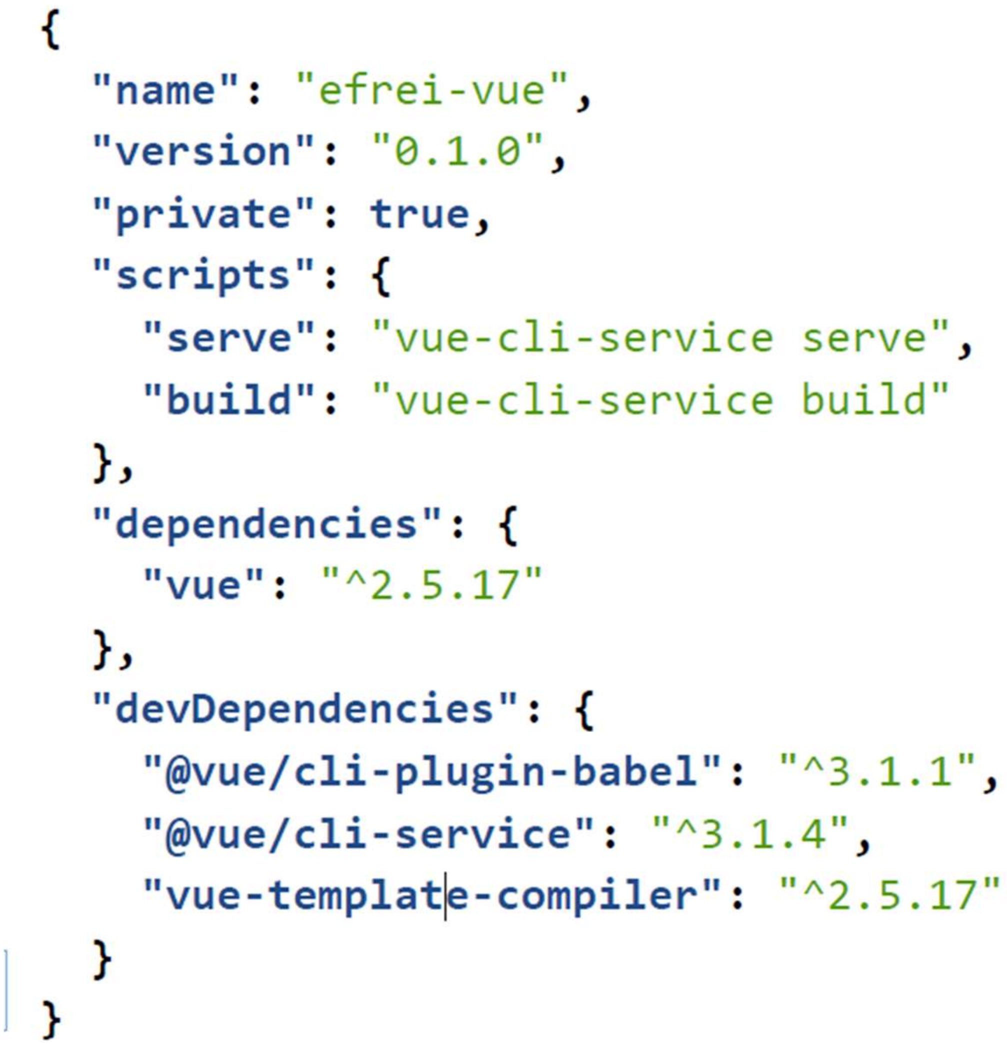
src/App.vue - behavior and state



```
<script>
import HelloWorld from './components/HelloWorld.vue'

export default {
  name: 'app',
  components: {
    HelloWorld
  }
}
</script>
```

Components can use other components!

package.json



```
{
  "name": "efrei-vue",
  "version": "0.1.0",
  "private": true,
  "scripts": {
    "serve": "vue-cli-service serve",
    "build": "vue-cli-service build"
  },
  "dependencies": {
    "vue": "^2.5.17"
  },
  "devDependencies": {
    "@vue/cli-plugin-babel": "^3.1.1",
    "@vue/cli-service": "^3.1.4",
    "vue-template-compiler": "^2.5.17"
  }
}
```

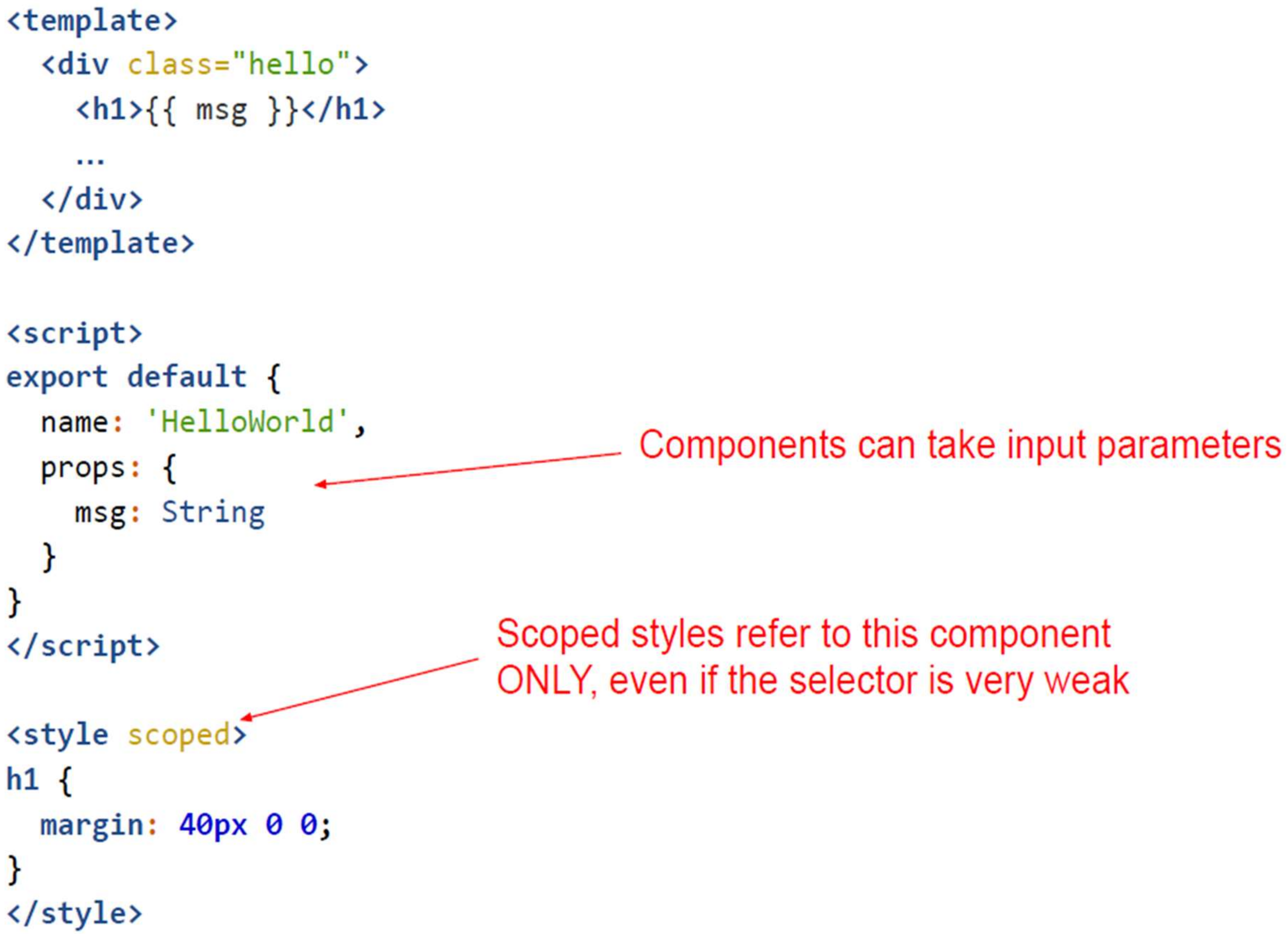
application name
version
private or public?

npm run serve script
npm run build script

for running,
project needs vue

and for development,
babel plugin
cli-service
and template compiler

src/components/HelloWorld.vue



```
<template>
  <div class="hello">
    <h1>{{ msg }}</h1>
    ...
  </div>
</template>

<script>
export default {
  name: 'HelloWorld',
  props: {
    msg: String
  }
}
</script>

<style scoped>
h1 {
  margin: 40px 0 0;
}
</style>
```

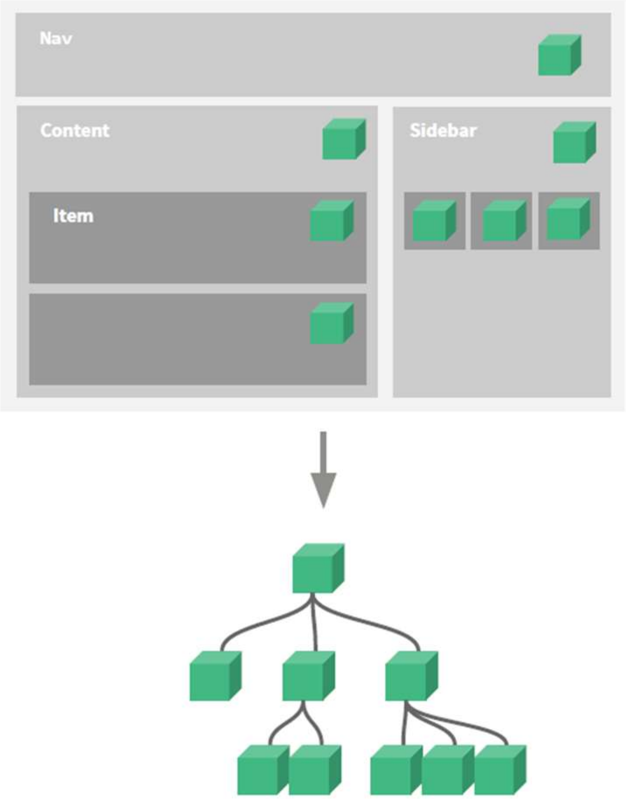
Components can take input parameters

Scoped styles refer to this component ONLY, even if the selector is very weak

Components aggregation

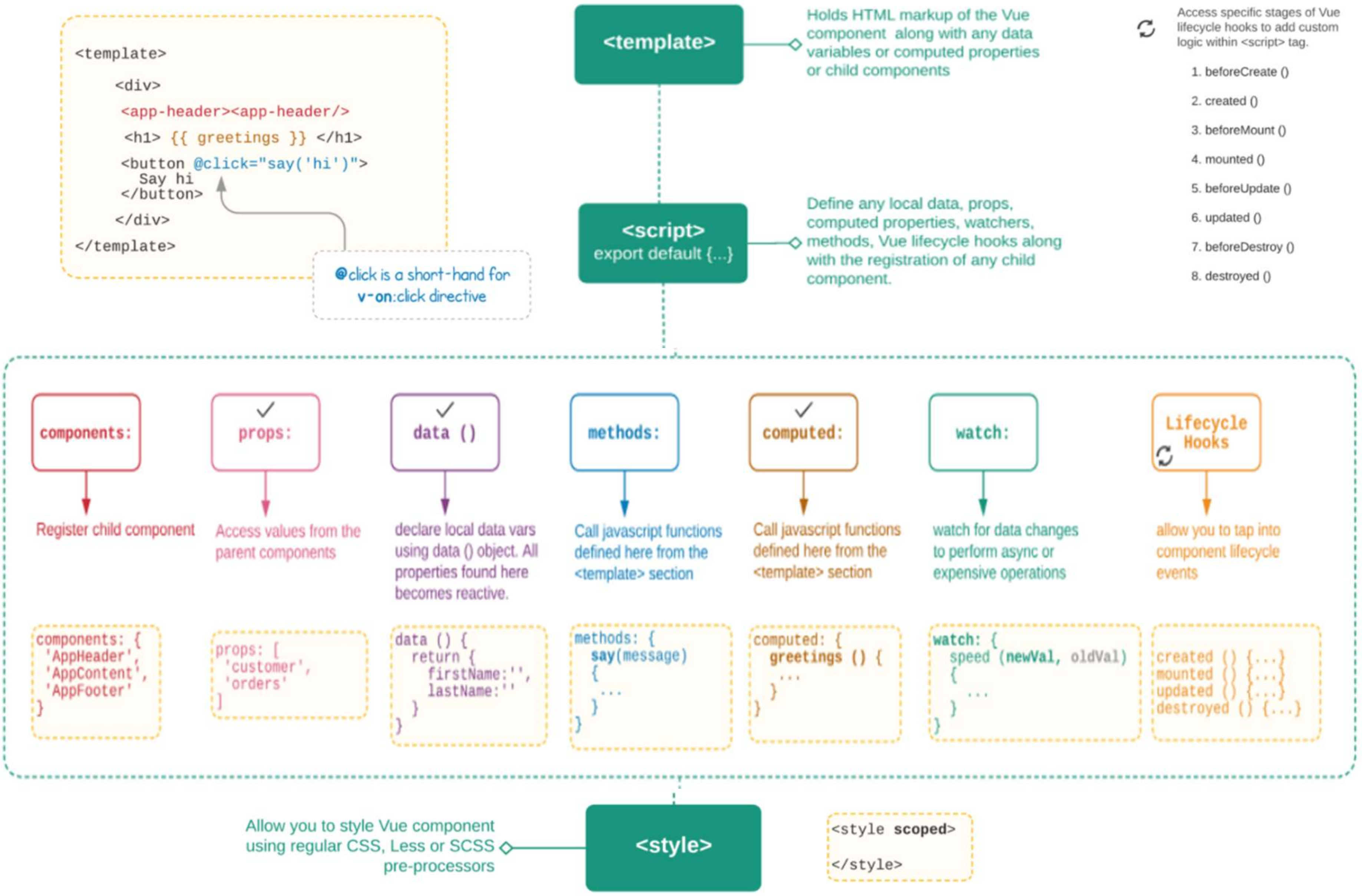
- components can use other components
- every component create a new HTML tag interpreted by the framework (Vue.js)
- components need to be imported and declared to be used in the template

```
<div id="app">
  <app-nav></app-nav>
  <app-content>
     <item></item>
     <item></item>
 </app-content>
 <sidebar>
 </sidebar>
</div>
```



app to be organized into a tree of nested components

Anatomy of Vue component



```
<template>

    <div>
     <app-header><app-header/>
     <h1> {{ greetings }} </h1>
     <button @click="say('hi')">
       Say hi
     </button>
    </div>
</template>
```

@click is a short-hand for v-on:click directive

Holds HTML markup of the Vue component along with any data variables or computed properties or child components

Define any local data, props, computed properties, watchers, methods, Vue lifecycle hooks along with the registration of any child component.

Access specific stages of Vue lifecycle hooks to add custom logic within <script> tag.

1. beforeCreate ()
2. created ()
3. beforeMount ()
4. mounted ()
5. beforeUpdate ()
6. updated ()
7. beforeDestroy ()
8. destroyed ()

Register child component

```
components: {
 'AppHeader',
 'AppContent',
 'AppFooter'
}
```

Access values from the parent components

```
props: [
 'customer',
 'orders'
]
```

declare local data vars using data () object. All properties found here becomes reactive.

```
data () {
  return {
    firstName:'',
    lastName:''
  }
}
```

Call javascript functions defined here from the <template> section

```
methods: {
  say(message)
  {
   ...
  }
}
```

Call javascript functions defined here from the <template> section

```
computed: {
  greetings () {
   ...
  }
}
```

watch for data changes to perform async or expensive operations

```
watch: {
  speed (newVal, oldVal)
  {
    ...
  }
}
```

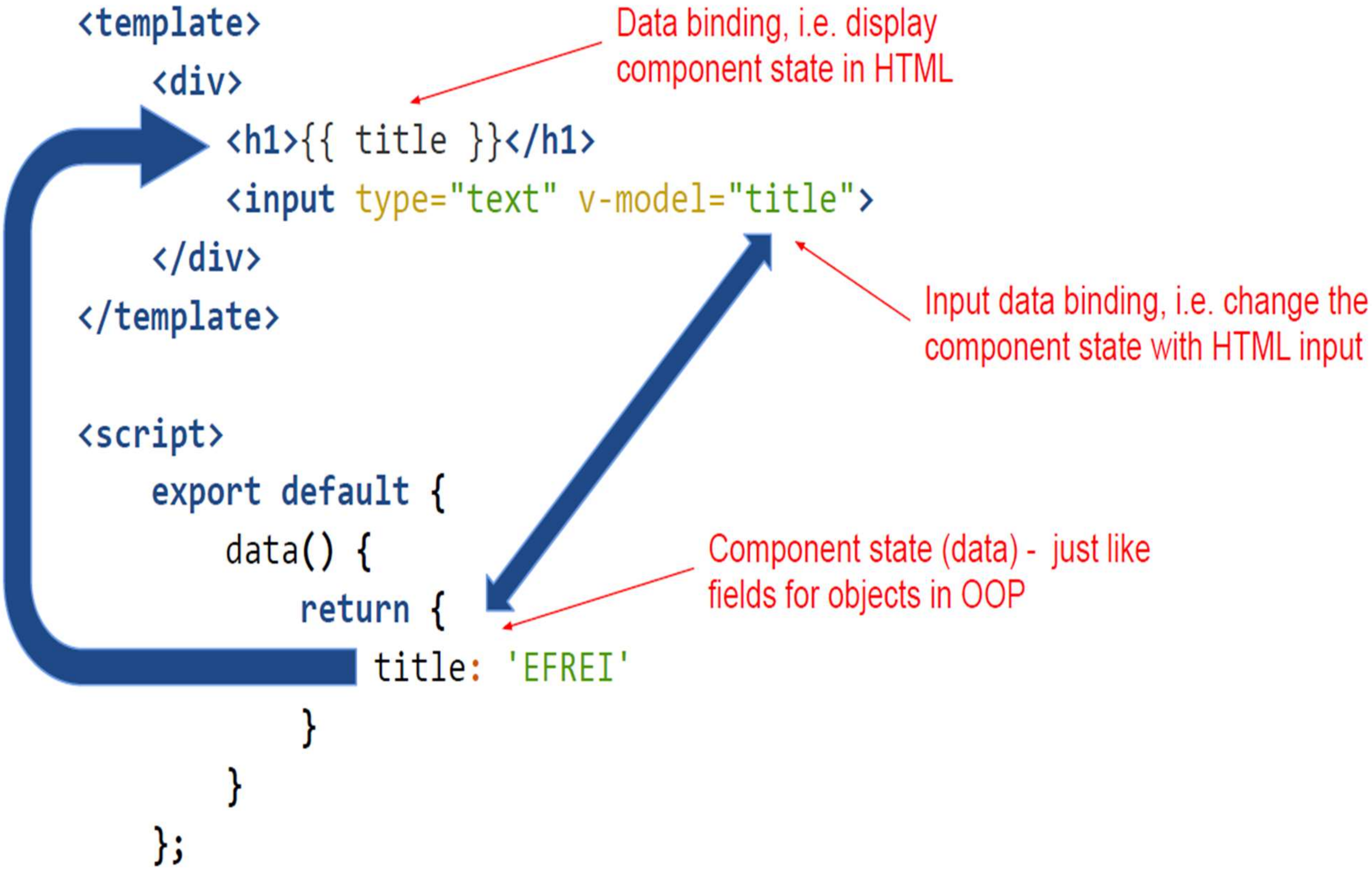
allow you to tap into component lifecycle events

```
created () {...}
mounted () {...}
updated () {...}
destroyed () {...}
```

Allow you to style Vue component using regular CSS, Less or SCSS pre-processors

```
<style scoped>

</style>
```

Reactivity in Components



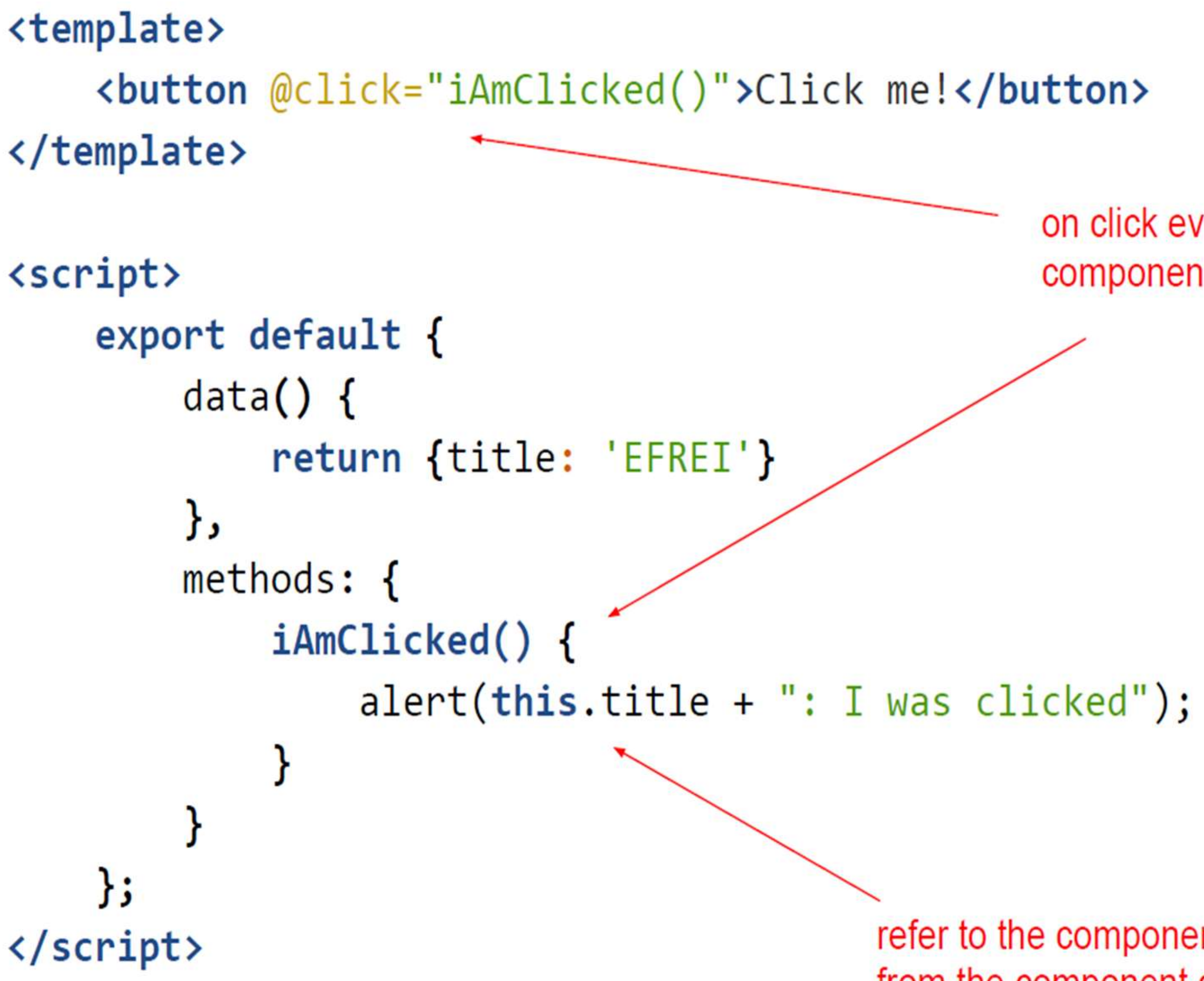
```
<template>
    <div>
        <h1>{{ title }}</h1>
        <input type="text" v-model="title">
    </div>
</template>

<script>
    export default {
        data() {
            return {
                title: 'EFREI'
            }
        }
    };
```

Data binding, i.e. display component state in HTML

Input data binding, i.e. change the component state with HTML input

Component state (data) - just like fields for objects in OOP

Component behavior

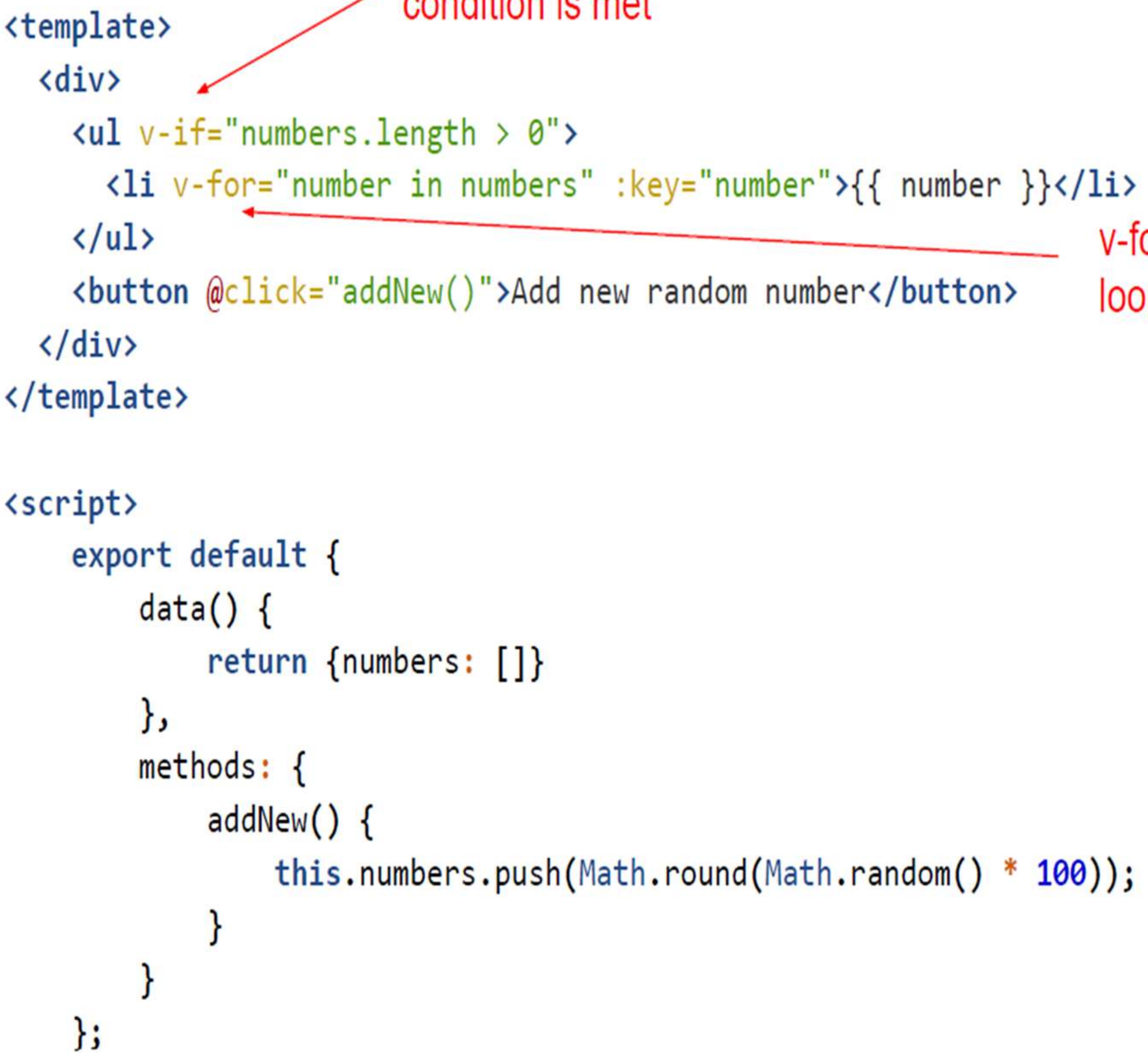


```
<template>
    <button @click="iAmClicked()">Click me!</button>
</template>

<script>
    export default {
        data() {
            return {title: 'EFREI'}
        },
        methods: {
            iAmClicked() {
                alert(this.title + ": I was clicked");
            }
        }
    };
</script>
```

on click event (@) call a component behavior (method)

refer to the component data/method from the component code

Directives



```
<template>
  <div>
    <ul v-if="numbers.length > 0">
      <li v-for="number in numbers" :key="number">{{ number }}</li>
    </ul>
    <button @click="addNew()">Add new random number</button>
  </div>
</template>

<script>
    export default {
        data() {
            return {numbers: []}
        },
        methods: {
            addNew() {
                this.numbers.push(Math.round(Math.random() * 100));
            }
        }
    };
```

v-if renders the element only if the condition is met

v-for renders elements in a loop (one for every value)

Lifecycle Hooks

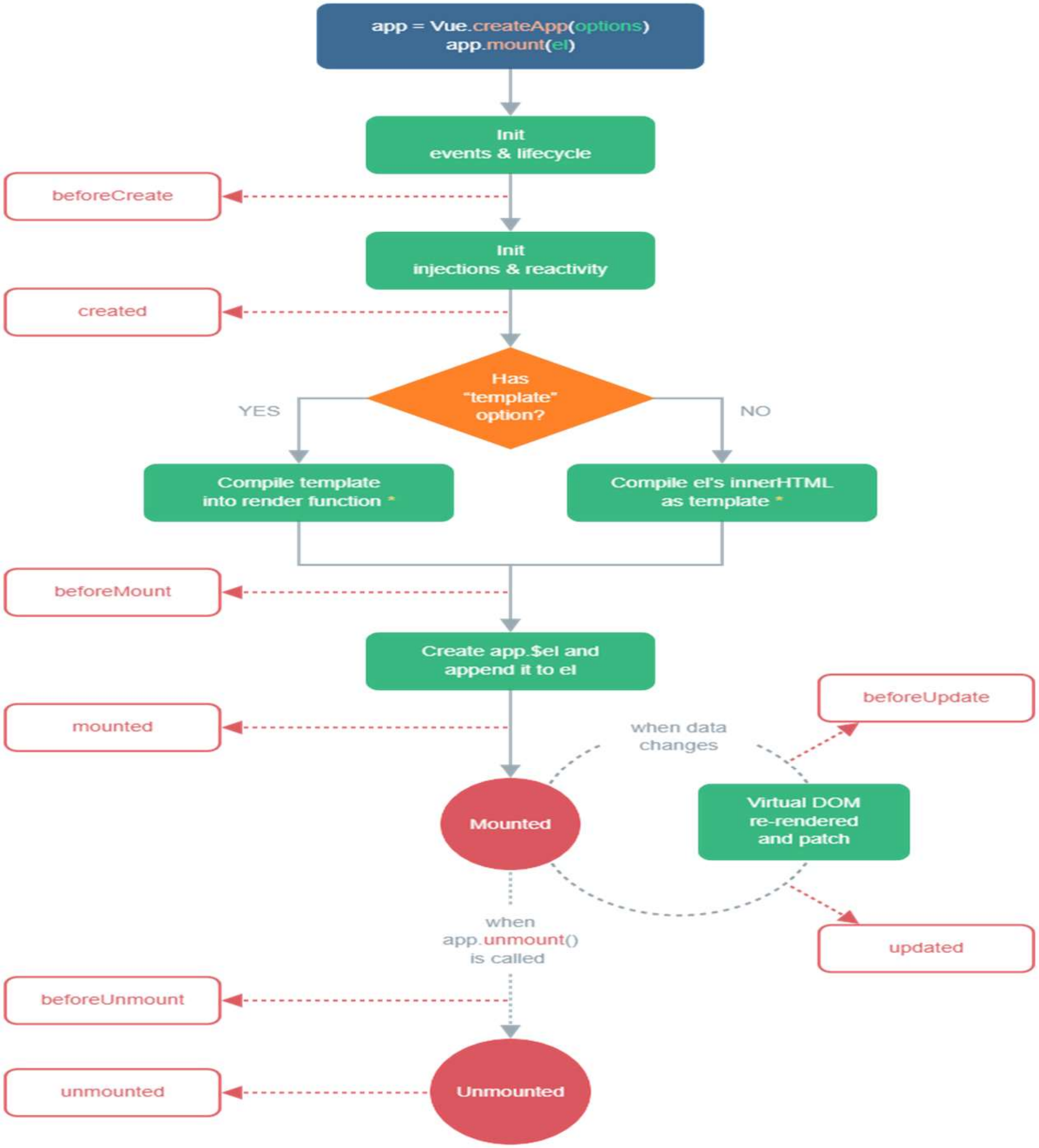
Each component instance goes through a series of initialization steps when it's created - for example, it needs to set up data observation, compile the template, mount the instance to the DOM, and update the DOM when data changes.

Along the way, it also runs functions called lifecycle hooks, giving users the opportunity to add their own code at specific stages.

the created hook can be used to run code after an instance is created:

```
Vue.createApp({
  data() {
    return { count: 1 }
  },
  created() {
        console.log('count is: ' + this.count)     // => "count is: 1"
  }
})
```

Lifecycle Diagram



component registration

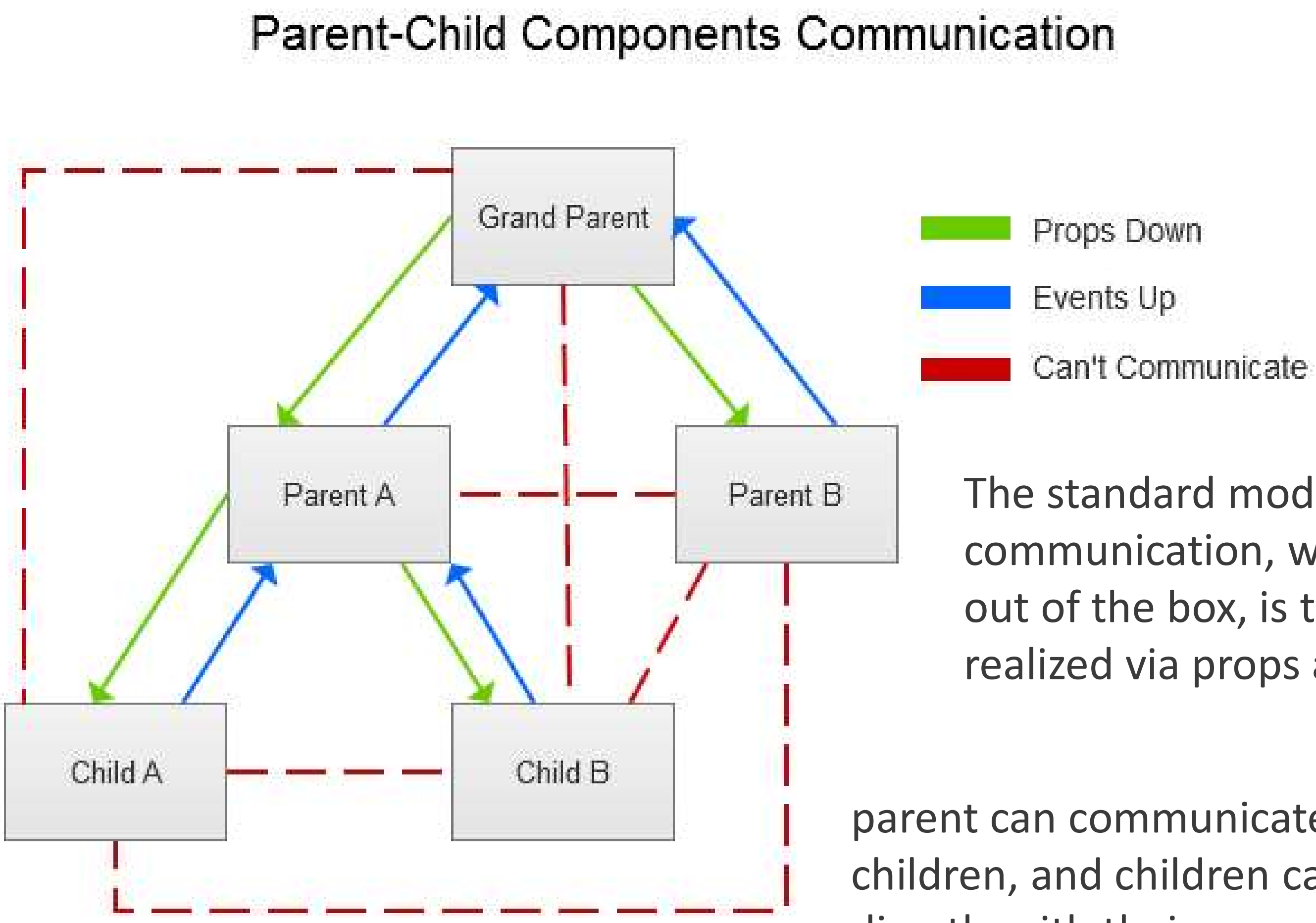
- To use components in templates, they must be registered so that Vue knows about them.
- There are two types of component registration: global and local.
- So far, we've only registered components **globally**, using the component method of our app:

```
const app = Vue.createApp({...})

App.component('my-component-name', {
        /* ... */
})
```

- Local registered:

```
const app = Vue.createApp({
  components: {
    'component-a': ComponentA,
    'component-b': ComponentB
  }
})
```

Direct Parent-Child Communication



The standard model of component communication, which Vue.js supports out of the box, is the parent-child model realized via props and custom events.

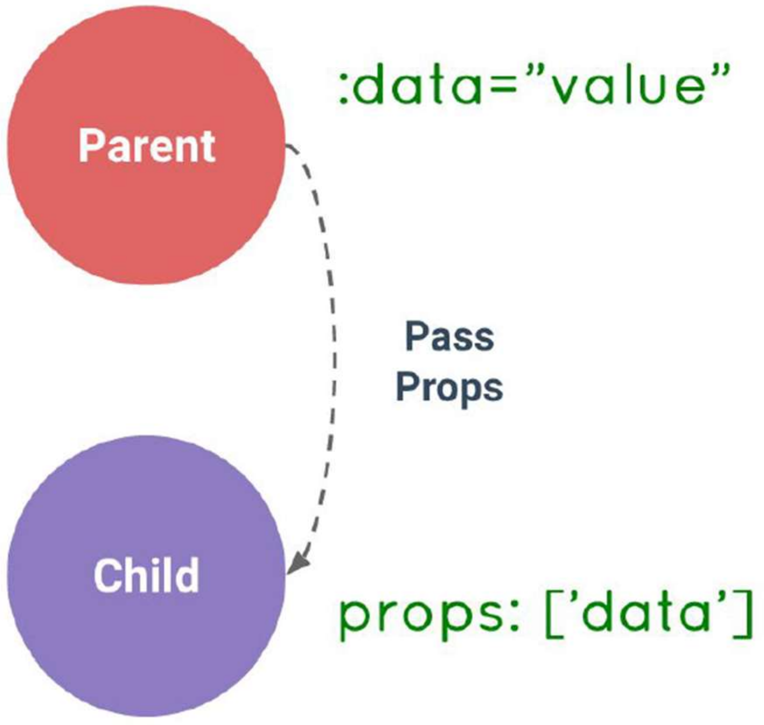
parent can communicate only with its direct children, and children can only communicate directly with their parent.
In this model, no sibling or cross-component communication is possible.

Passing Data to Child Components with Props

Props are used to pass down state to child components.

```
<div id="blog-post-demo" class="demo">
  <blog-post title=" Title A"></blog-post>
  <blog-post title=" Title B "></blog-post>
  <blog-post title=" Title C "></blog-post>
</div>
```

```
app.component('blog-post', {
    props: ['title'],
  template: `<h4>{{ title }}</h4>`
})
```



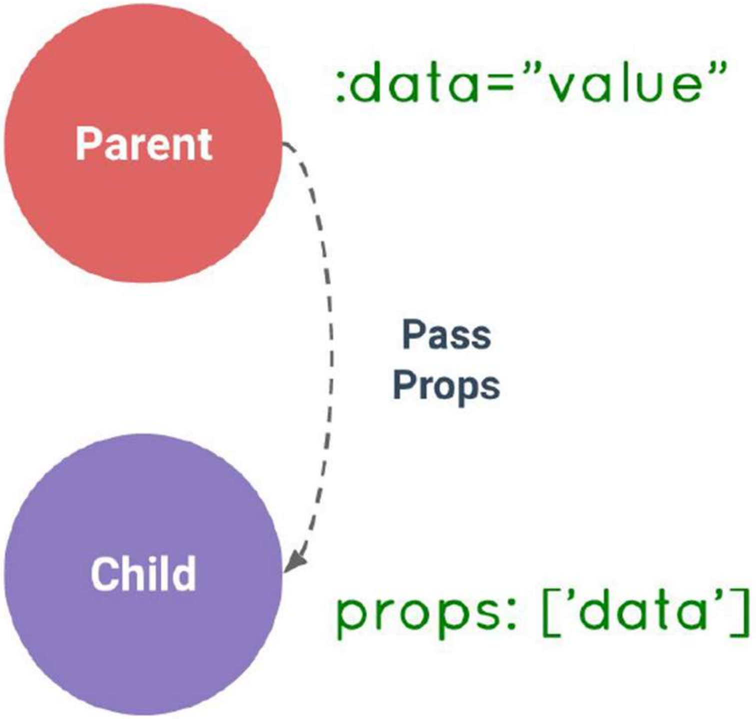
Parent-to-Child Communication

To dispatch data from a parent to its children, Vue.js uses props. There are three necessary steps to pass down a property:

1. Registering the property in the child, like this: props: ["score"]
2. Using the registered property in the child's template, like this: <span>Score: {{ score }}</span>
3. Binding the property to the score variable (in parent's template), like this: <child-a :score="score"/>

Passing Data to Child Components with Props part 2

```
const App = {
  data() {
    return {
      posts: [
        { id: 1, title: 'My journey with Vue' },
        { id: 2, title: 'Blogging with Vue' },
        { id: 3, title: 'Why Vue is so fun' }
      ]
    }
  }
}
const app = Vue.createApp(App)

app.component('blog-post', {
  props: ['title'],
  template: `<h4>{{ title }}</h4>`
})
app.mount('#blog-posts-demo')
```



Validating Props

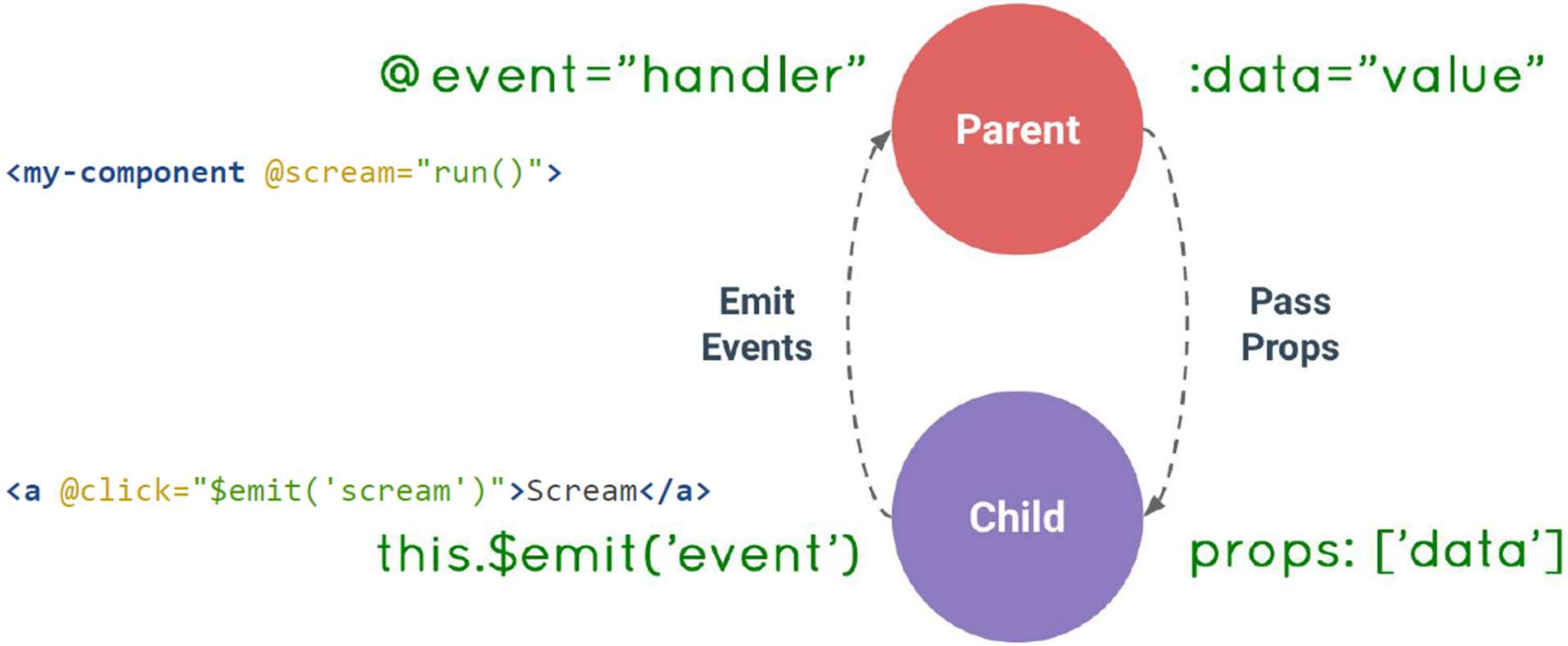
it's recommended to validate the props. This will ensure that the props will receive the correct type of value. For example, our score property could be validated like this:

```
props: {
    // Simple type validation
    score: Number,

  // or Complex type validation
    score: {
      type: Number,
      default: 100,
      required: true
    }
  }
```

When prop validation fails, Vue will produce a console warning

Listening to Child Components Events

Events allow you to communicate from the children up to the parent:



```
<my-component @scream="run()">
```

```
<a @click="$emit('scream')">Scream</a>
```

Child components communicate with their parents by using events: they emit events that propagate from parent to parent, in the same way as DOM events like a mouse click

Custom events

```
// fire custom event
this.$emit("eventName", data);
```

```
<!--
$event == event data
when _eventName_ event happens, call
_functionName_ function
-->
<p v-on:eventName="functionName($event)"></p>
```

Listening to Child Components Events

Events allow you to communicate from the children up to the parent:

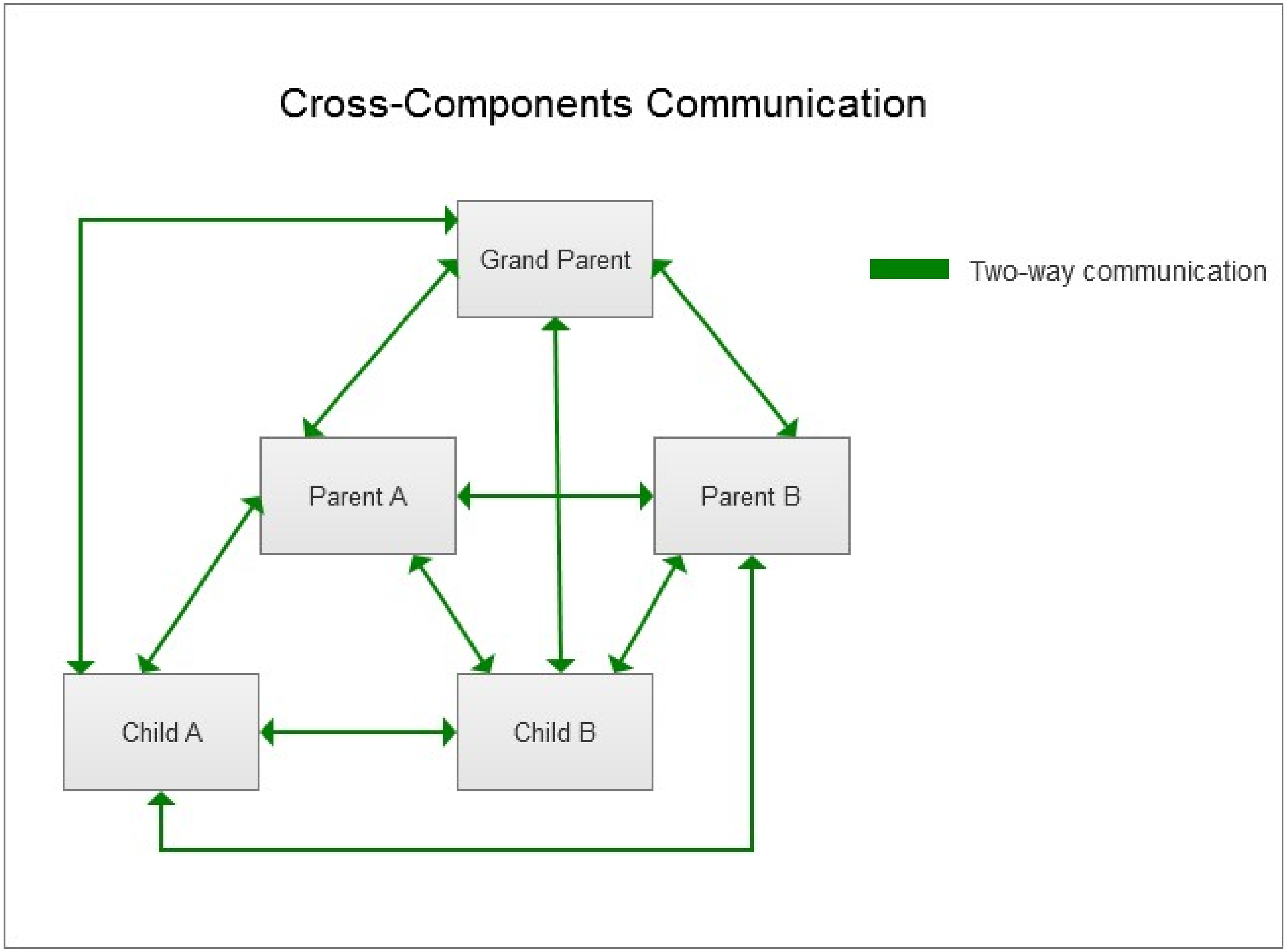
children

```
<script>
export default {
  name: 'Car’,
  methods: {
    handleClick: function() {
       this.$emit('clickedSomething’)
    }
  }
}
</script>
```

parent

```
<template>
<div>
<Car v-on:clickedSomething="handleClickInParent" />
<!-- or -->
<Car @clickedSomething="handleClickInParent" />
</div>
</template>

<script>
export default {
name: 'App',
methods: {
   handleClickInParent: function() {
     //...
} } }
</script>
```

Cross-Component Communication



Communication

Components in Vue.js out of the box can communicate using

props, to pass state down to child components from a parent

events, to change the state of a parent component from a child, or using the root component as an event bus

Using an Event Bus to communicate between any component

- Using events you're not limited to child-parent relationships.
- You can use the so-called Event Bus.
- Above we used this.$emit to emit an event on the component instance.

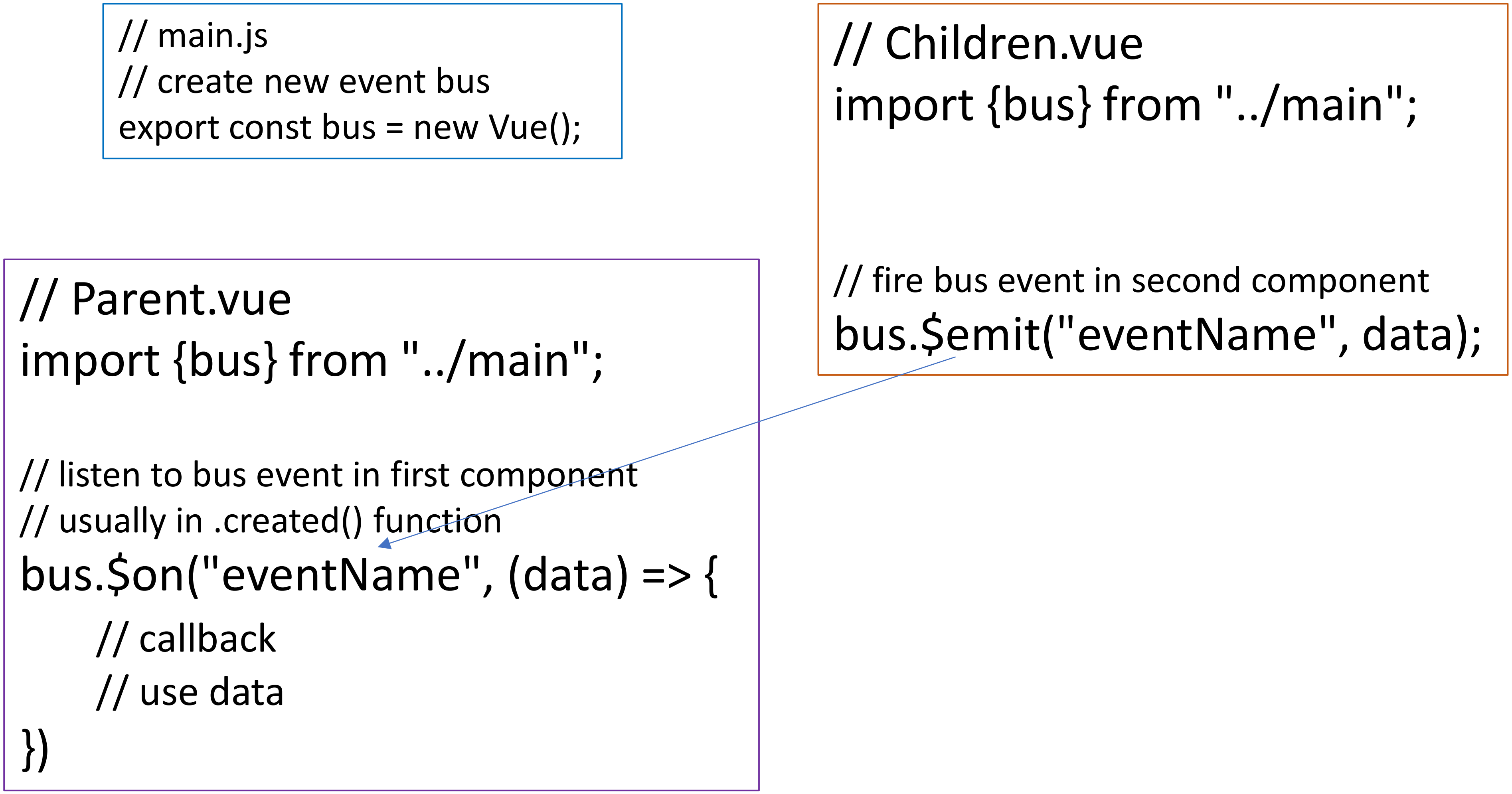
You can just send the data from one component using
this.$root.$emit('name-of-emitter', args1, args2, ...)

and is captured using the same name like this
this.$root.$on('name-of-emitter', args1, args2, ...)

in the other component.

Alternatives: Vuex - official state management library for Vue.js

Using an Event Bus to communicate between any component

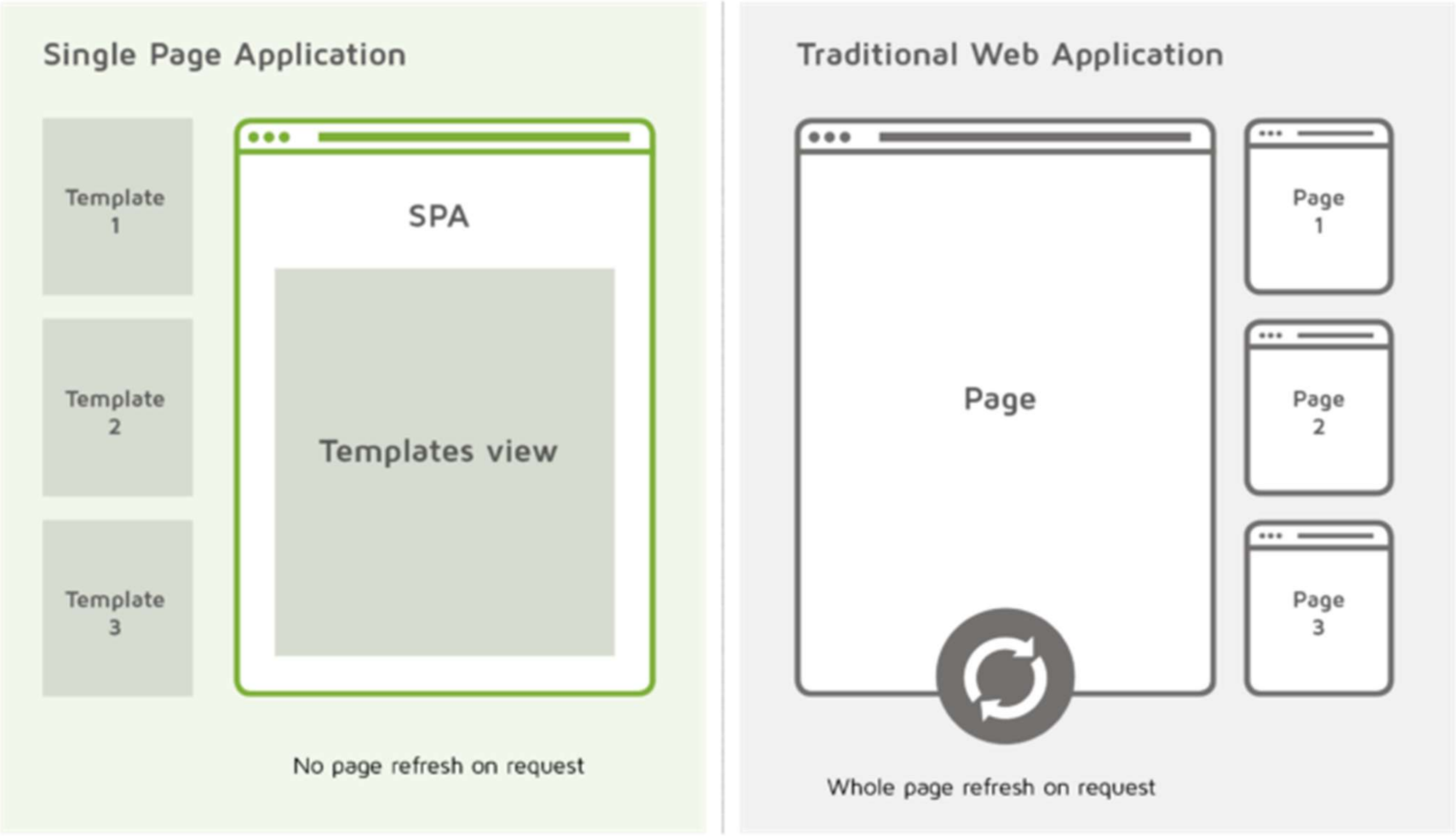


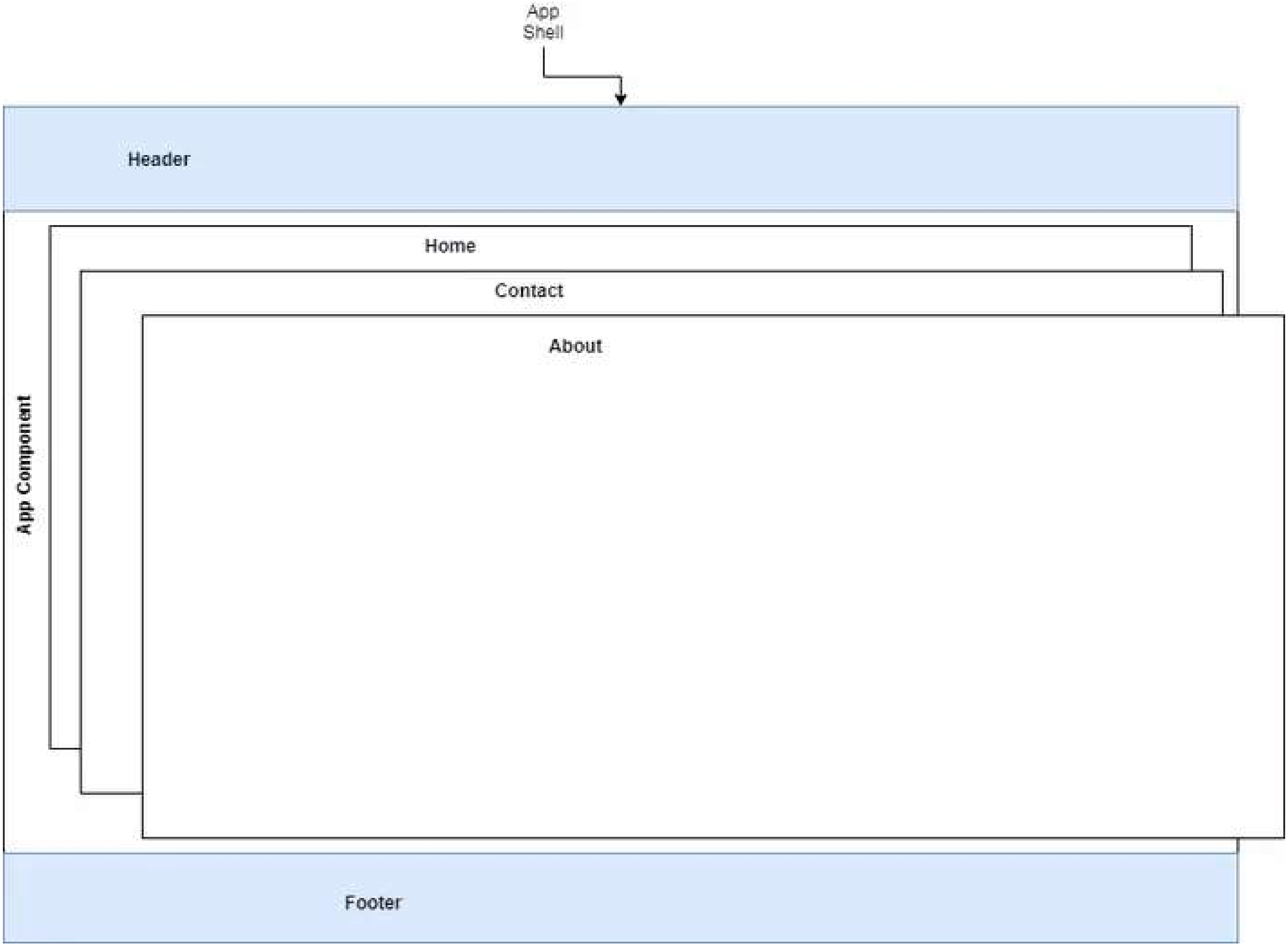
```
// main.js
// create new event bus
export const bus = new Vue();
```

```
// Children.vue
import {bus} from "../main";


// fire bus event in second component
bus.$emit("eventName", data);
```

```
// Parent.vue
import {bus} from "../main";

// listen to bus event in first component
// usually in .created() function
bus.$on("eventName", (data) => {
	// callback
	// use data
})
```

Routing – navigation through the application




App Shell
Header
Home
Contact
About
App Component
Footer

Routing with vue-router

- Vue applications are mostly Single Page Applications (SPA). The server always serves a single HTML page, and navigation between the application pages/sections is managed on the client side in JavaScript. This approach allows smoother transitions between pages, and reduces the number of server calls needed to navigate between pages.

A library for this purpose: vue-router. This router allows you to associate routes (URLs) with Vue components, and offers many features:

- Routes tree configuration
- Modular configuration based on components
- Dynamic parameters handling: path, query, wildcards...

Instalation

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

```
import router from "./router";

createApp(App).use(router).mount("#app")
```

After you call use(router) passing the router object, in any component of the app you have access to these objects:

- this.$router is the router object
- this.$route is the current route object

Router Configuration

The router is created by taking a list of routes as parameters. Each route associates a URL pattern with a certain component. When the page loads, or when the URL changes, the router will resolve which route is associated with this new URL.

```
** src/router/index.js **/
import { createRouter, createWebHashHistory } from 'vue-router'
import HelloWorld from "@/components/HelloWorld";

export default createRouter({
  history: createWebHashHistory(),
  routes: [
    {
      path: "/hello/:name",
      name: "hello",
      component: HelloWorld
    }
  ]
})
```

Navigation and router-link

Vue-router includes a globally declared <router-link> component, which can substitute <a> tags for any internal navigation done via this router.

The advantage of this component over traditional <a> tags is that the links will always match your configuration (hash or history) and can be static or dynamically generated by route names and parameter lists:

```
<template>
<div id="app">
<nav>
   <router-link to="/">Home</router-link>
   <router-link to="/login">Login</router-link>
   <router-link to="/about">About</router-link>
</nav>
<router-view></router-view>
</div>
</template>
```

Vue-router also brings methods to all components to programmatically navigate between pages:

```
this.$router.go(-1); // go to previous page
let nextId = this.$route.params.id + 1; // get URL path param
this.$router.push(`/article/${nextId}`); // navigate to a new page by URL
```