LECTURE 8: THICK CLIENT APPLICATIONS - VUE.JS - I

MOHAMAD GHARIB
UNIVERSITY OF TARTU
Spring Boot Server - recap
Spring Boot Server - recap

Client

HTTP request

HTTP response

Spring Boot Server

Entity | Controller | Service | Repository (JPA)

Database
Spring Boot Server - recap

Spring Boot Server

Client

HTTP request

HTTP response

Entity
Controller
Service
Repository (JPA)
The Client side

**GET** /plants

HTTP request

HTTP response

Spring Boot Server

plantsRepository.findAll()
The Client side

Client

GET /plants/id

Spring Boot Server

plantsRepository.findById(id)

Entity

Controller

Service

Repository()
The Client side

Client

POST /plants

HTTP request

 plantsRepository.save(plant)

HTTP response

Spring Boot Server
The Client side

Client

PUT /plants/id
DELETE /plants/id

HTTP request

HTTP response

Spring Boot Server

plantsRepository.save(plant)
plantsRepository.delete(id)
Vue.js
**Vue.js**

**Vue** is a progressive **framework** for building user interfaces. **Vue** is a front-end Javascript framework. **Vue** is capable of powering sophisticated **Single-Page Applications (SPA)**, when used in combination with modern tooling and supporting **libraries**.

**Vue** can be used also to create **Standalone wedges**

- Light weight
- Routing
- Reusable components
- Directives
- Event Handling
- Watched and computed properties
- Unit and Integration testing
- Command Line Interface (CLI)
Single-Page Applications (SPA)?
Client-side vs. Server-side rendering
In **server-side** rendering, the server **compiles** everything and **delivers** a fully populated HTML page to the **client**. This is done **every time** you navigated to a new route/page.
In **client-side** rendering, the **server** will deliver a single HTML file without “any content” to the **client**. Then, the **client** fetches and compiles everything before **rendering** the content.
Vue.js - Single-Page Applications (SPA)

Client

Bare-bones HTML document

Server

initial request

Bare-bones HTML document
Vue.js - Single-Page Applications (SPA)
Vue.js - Single-Page Applications (SPA)
Vue.js - Single-Page Applications (SPA)

Client

- Header component
- Body component
- Side component
- Footer component

Server

- link clicked
- new component rendered
- Vue.js bundle

Database
Vue.js in nutshell
Vue.js in nutshell

![Vue.js diagram]

- **Vue.js**
  - Home | Plants | About
  - Header component
  - Body component
  - Side component
  - Footer component

- **Server**

---

21
Vue.js in nutshell

Vue routes and Views

A view is supposed to contain root level component(s), where other components would be imported.
Vue.js in nutshell

Vue routes and Views
A view is supposed to contain root level component(s), where other components would be imported.

Components
Components are reusable Vue instances with custom HTML elements. Components can be reused as many times as you want or used in another component, making it a child component. Data, computed, watch, and methods can be used in a Vue component.

Server
Vue.js in nutshell

Vue routes and Views
A **view** is supposed to contain root level **component(s)**, where other components would be imported.

Components
- Component structure
  - `<template>`
  - `<script>`
  - `<style>`

How we can use components?
- Methods declaration and use
- Data declaration and use
- Computed & Watched properties
- Passing data to components (props: {})
- Props type checks and other validators

Directives
A **directive** is a special token in the markup that tells the library to attach a specified behavior to that DOM element.
Vue CLI project
Vue CLI project

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  router  vuex  eslint

Essential Links

Core Docs  Forum  Community Chat  Twitter  News

Ecosystem

vue-router  vuex  vue-devtools  vue-loader  awesome-vue

DONE  Compiled successfully in 8463ms
- Local:  http://localhost:8080/
- Network:  http://192.168.1.147:8080/

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

The **node_modules** directory contains the full set of dependencies for the application.

The **package.json** file and this is how the application knows which `node_modules` to get.

The **.gitignore** contains files that will not be committed to the repo during git push or commit operations.

The **README.md** basic readme file that contains essential commands to install and run your project.
Vue CLI project

src directory contains:
- assets
- components
  - HellowWorld.vue
- router
  - index.js
- views
  - About.vue
  - Home.vue
- App.vue
- main.js
Vue Routers & Views
1. Create a View

- In the Views directory, create a new file (e.g., Plants.vue).
- Export the view.
Views & Vue Router

1. Create a View
   • In the Views directory, create a new file (e.g., Plants.vue).
   • Export the view.

2. Create a Vue Route
   • In the route/index.js, update the routes array by adding the new route.
   • Register/Import the new view.
Views & Vue Router

1. Create a View
   - In the Views directory, create a new file (e.g., Plants.vue).
   - Export the view.

2. Create a Vue Route
   - In the route/index.js, update the routes array by adding the new route.
   - Register/Import the new view.
Views & Vue Router

1. Create a View
   • In the Views directory, create a new file (e.g., Plants.vue).
   • Export the view.

2. Create a Vue Route
   • In the route/index.js, update the routes array by adding the new route.
   • Register/Import the new view.

3. Create the router-link
   • In App.vue, add the route link.

   `<router-view />` is a functional component that renders the matched component for the given path.
1. Create a View
   • In the Views directory, create a new file (e.g., Plants.vue).
   • Export the view.

2. Create a Vue Route
   • In the route/index.js, update the routes array by adding the new route.
   • Register/Import the new view.

3. Create the router-link
   • In App.vue, add the route link.

To test how SPA works: In the browser, Right-click -> inspect open the network tab and try to navigate between the Vue app routes. You should notice that there are no traffic.
Components

Single File Components (SFC)
**Components**

Components (SFC) are reusable Vue instances with a name. We can use this component as a custom element inside a root Vue instance.
Components (SFC) are reusable Vue instances with a name.

We can use this component as a custom element inside a root Vue instance.

A Vue component is a special file format that allows us to encapsulate the template, logic, and styling in a single file.
Components

1. Create a component
   - In `components` directory, create a new file (e.g., `PlantCompo.vue`).
   - Add the structure `<template>`, `<script>`, and `<style>`.

2. Export the component, i.e., make it available to be used by other components and views.
   ```javascript
   export default {
     name: "PlantCompo",
     props: {
       msg: String,
     },
   };
   ```
Components

1. Create a component
   • In components directory, create a new file (e.g., PlantCompo.vue).
   • Add the structure `<template>`, `<script>`, and `<style>`.  
2. Export the component, i.e., make it available to be used by other components and views.
   ```
   export default {name: “PlantCompo”, }
   ```
3. Register/Import the new component in the parent component/view.
4. The newly created component can be used in another component or view as a tag using the key it is registered under.
   ```
   <PlantCompo />
   <plant-compo> </plant-compo>
   ```
Components

1. Create a component
   - In components directory, create a new file (e.g., PlantCompo.vue).
   - Add the structure `<template>`, `<script>`, and `<style>`. 

2. Export the component, i.e., make it available to be used by other components and views.
   ```javascript
   export default {name: "PlantCompo", } 
   ```

3. Register/Import the new component in the parent component/view.

4. The newly created component can be used in another component or view as a tag using the key it is registered under.
   ```html
   <PlantCompo />
   </plant-compo> </plant-compo>
   ```
Components - <template>

<template> is an HTML-based template that allows binding the rendered DOM to the underlying component instance’s data.

<template> contains valid HTML that can be parsed by browsers and HTML parsers.

To take full advantage of the <template>, we need to understand:

• Data declaration and use (Data type checks and other validators)
• Passing data to components (props: {})
• Computed & Watched properties
• Directives.

HTML Tutorial: https://www.w3schools.com/html/
each *.vue file may contain a *<script>* block.

*<script>* contains the “data” and most of the logic () in the component, such as:

- Data declaration (Data type checks and other validators)
- Passing data to components (props: {})
- Computed & Watched properties
- Methods
- Lifecycle Hooks.
Components - <style>

<style> each *.vue file may contain a <style> block.

<style> contains the “styling” of the component In components, which is written in CSS.

CSS is the language that can be use to style an HTML document.

<style scoped> is used to limited the application of the defined styling to the component.

CSS Tutorial: https://www.w3schools.com/css/
Components - `<style>`

Each `.vue` file may contain a `<style>` block.

`<style>` contains the “styling” of the component in components, which is written in CSS.

CSS is the language that can be used to style an HTML document.

`<style scoped>` is used to limit the application of the defined styling to the component.

keep an eye on the `<style>`

CSS Tutorial: https://www.w3schools.com/css/
Data, Props and methods
Data must be declared as a function that returns the data object(s).

Why? because if we declared data as an object, it will be shared by reference across all created component instances.

By providing a data function, every time a new instance is created we can call it to return a fresh copy of the initial data.

The most basic form of data binding is text interpolation using the “Mustache”`{{}}`, which can be also used for Using JavaScript Expressions.

Data declaration and use

Data must be declared as a function that returns the data object(s).

Why? because if we declared data as an object, it will be shared by reference across all created component instances.

By providing a data function, every time a new instance is created we can call it to return a fresh copy of the initial data.

The most basic form of data binding is text interpolation using the “Mustache” `{{}}`, which can be also used for Using JavaScript Expressions.

**Props**: `{ }`

Props are used to indicate what external data should be passed to the component.

**Props vs Data**: props are meant to be propagated (passed) and managed from parent components, while data is the component internal state, i.e., the component is responsible for.

**Props types**: props have types (e.g., String, Number, Boolean, Array, Object, Date, Function).
**Props**: `{ }`

**Props** are used to indicate what external data should be passed to the component.

**Props vs Data**: props are meant to be propagated (passed) and managed from parent components, while data is the component internal state, i.e., the component is responsible for.

**Props types**: props have types (e.g., String, Number, Boolean, Array, Object, Date, Function).
Props checks and other validators

Components can specify requirements for their props:

Types checks, Vue checks if a prop has a given type, and will throw a warning if it does not.

Props checks and validators:

Default, If a default value is specified, it will be used if the resolved prop value is undefined.

Required, All props are optional by default, unless required: true is specified.

Custom validator function that takes the prop value as the sole argument.

Note: when prop validation fails, Vue will produce a console warning.

https://vuejs.org/guide/components/props.html#prop-validation
Methods to be mixed into the Vue instance. You can access these methods directly on the VM instance, or use them in directive expressions.

All methods will have their `this` context automatically bound to the Vue instance.
Directives

A directive is a special token in the markup that allows attaching a specified behaviour to that DOM elements.

https://vuejs.org/api/built-in-directives.html
Directives: v-text & v-html

**v-text:** updates the element's text content, so it will overwrite any existing content inside the element.

```html
<span v-text="msg"></span>
```

**v-text vs. {}{},** if you need to update a part of text content, it is better to `{{}}`.

**v-html:** updates the element's innerHTML, and it expects String.

```html
<p v-html="messageHtml"></p>

<!-- messageHtml: "<h1> Hi I am an HTML<h1>" -->
```

Hi, I am a v-text

Hi, I am a v-HTML
Directives: v-on

**v-on** *(Shorthand: “@”)* Attaches an event listener to an element. The event type is denoted by the argument. The expression can be a method or an inline statement.

```html
<!-- method handler -->
<button v-on:click="dosomething"></button>

<!-- inline statement -->
<button v-on:click=" dosomething('hello', $event)"></button>
```

Hello John, you have clicked 7 times.

```html
<!-- v-on -->
<p @click='count++'> Hello {{nameClick}}, you have clicked {{count}} times </p>
```
**Directives: v-show & v-once**

**v-show**: toggle the element's visibility based on the truthy-ness of the expression value.

*v-show* works by setting the display CSS property via inline styles, and will try to respect the initial display value when the element is visible. It also triggers transitions when its condition changes.

```html
<h1 v-show="true">Hello!</h1>
<h1 v-show="count>4"> count: {{count}}</h1>
```

**v-once**: render the element and component once only, and skip future updates.

On subsequent re-renders, the element/component and all its children will be treated as static content and skipped.

```html
<plant-compo v-once msg = "This is a plant component"
```

**v-show test count: 7**

Appear if random number bigger than 0.4

```html
<!-- <v-show> -->
<h1 v-show="true">Hello!</h1>
<h1 v-show="count>4"> v-show test count: {{count}}</h1>
```
Directives: v-if, v-else-if & v-else

**v-if:** conditionally render an element or a template fragment based on the truthy-ness of the expression value.

```
<p v-if="Math.random() > 0.4"> Appear if random number bigger than 0.4 </p>
```

**v-else-if:** Restriction: previous sibling element must have v-if or v-else-if.

```
<p v-else-if="Math.random() == 0.4"> Appear if random number equal to 0.4 </p>
```

**v-else:** denote the "else block" for v-if or a v-if / v-else-if chain.

```
<p v-else> "random number not bigger than 0.4" </p>
```
Directives: v-bind & v-model

**v-bind (Shorthand "":"**): dynamically bind one or more attributes, or a component prop to an expression.

```html
<!-- bind an attribute -->
<img v-bind:src="imageSrc">
<!-- shorthand -->
<input type = "text" :value="writeSomething">
```

**v-model**: create a two-way binding on a form input element or a component.

```html
<input type = "text" v-model ="writeSomething">
```
**Directives: v-for**

**v-for**: render the element or template block multiple times based on the source data.

```html
<div v-for="plant in plants"> {{ plant.id }}</div>
```
Computed and watched properties

https://vuejs.org/guide/essentials/computed.html
https://vuejs.org/guide/essentials/watchers.html
**Computed properties**

**computed properties** enable you to create a property that can be used to modify, manipulate, and display data within your components in a readable and efficient manner.

**computed properties** is a way to reduce the complexity of the `<template>` as it can become bloated and hard to maintain.

Unlike **methods**, **computed properties** will be “re-calculated” anytime some data changes in the component.
Computed properties

**Computed properties** enable you to create a property that can be used to modify, manipulate, and display data within your components in a readable and efficient manner.

**Computed properties** is a way to reduce the complexity of the `<template>` as it can become bloated and hard to maintain.

Unlike **methods**, **computed properties** will be “re-calculated” anytime some data changes in the component.
Watched properties (Watchers)

**Watchers** allow the developers to listen to the component data and run whenever they change a particular property.

**Watcher** is a unique Vue.js feature that lets you keep an eye on one property of the component state and run a function when that property value changes.
Lifecycle hooks

https://vuejs.org/api/options-lifecycle.html
Lifecycle hooks

new Vue()

- Init Events & Lifecycle
  - beforeCreate
  - created

- Init injections & reactivity
  - Has 'el' option?
    - NO
      - when vm.$mount() is called
    - YES
      - Has 'template' option?
        - NO
          - Compile el's outerHTML as template
        - YES
          - Compile template into render function

beforeMount

Create vm.$el and replace 'el' with it

mounted

when data changes

beforeUpdate

Mounted

Virtual DOM re-render and patch

when vm.$destroy() is called

beforeDestroy

Tear down watchers, child components and event listeners

destroyed

Destroyed

64
Lifecycle hooks

beforeCreate
Called synchronously immediately after the instance has been initialized, before data observation and event/watcher setup.

created
Called synchronously after the instance is created
At this stage:
data observation, computed properties, methods, watch/event callbacks have been set up.
The mounting phase has not been started, and the $el property will not be available yet.
**Lifecycle hooks**

**beforeMount**
Called right before the mounting begins; the render function is about to be called for the first time.

**mounted**
Called after the instance has been mounted, where `el` is replaced by the newly created `vm.$el`. If the root instance is mounted to an in-document element, `vm.$el` will also be in-document when mounted is called.

[Diagram showing lifecycle hooks with explanations for each step.]
Lifecycle hooks

**beforeUpdate**
Called when data changes, before the DOM is patched. This is a good place to access the existing DOM before an update.

**updated**
Called after a data change causes the virtual DOM to be re-rendered and patched. The component’s DOM will have been updated when this hook is called, so you can perform DOM-dependent operations here. However, in most cases you should avoid changing state inside the hook.
Lifecycle hooks

**beforeCreate**
Called right before a Vue instance is created. This stage happens very early and before any actual data or logic has been processed.

**created**
Called right after the instance has been created. The instance is now fully functional and ready to receive data and user interactions.

**beforeMount**
Called right before the instance is mounted. This stage happens before the virtual DOM is inserted into the real DOM.

**mounted**
Called right after the instance is mounted. The instance is now fully rendered and interactive.

**beforeUpdate**
Called right before a virtual DOM update. This stage happens after the data changes but before the update is rendered.

**updated**
Called right after a virtual DOM update. This stage happens after the data changes are applied to the virtual DOM.

**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. When this hook is called, all directives of the Vue instance have been unbound, all event listeners have been removed, and all child Vue instances have also been destroyed.
Lifecycle hooks - example

mounted() is commonly used for fetching data from “external sources” to be used in mounted elements.
NPM
Node Package Manager
npm\textsuperscript{1} is a package manager for the JavaScript programming language.

\textit{npm} has its own command-line interface (CLI).

\textit{npm} offers a registry that is an online database of public and paid-for private package.

Why \textit{npm} is very important: it allows adapting packages of code for your apps, or incorporating packages as they are.

\textsuperscript{1}https://www.npmjs.com/
Thank You for your attention

Mohamad Gharib
mohamad.gharib@ut.ee

UNITRe OF TARTU

unitartu
tartuuniversity