Web Application Development
(LTAT.05.004)

VUE.JS - I

MOHAMAD GHARIB
UNIVERSITY OF TARTU
JavaScript - Recap

1993
Mosaic

1994
Netscape

1995
IE

1995
Mocha → liveScript → JS
Brendan Eich at Netscape

1998
ECMAScript 2 (ES2)

2000-2008
ES4 - Never released

2000-2005
AJAX

2001
JSON

2015
ES6 / ECMAScript 2015

2016
ECMAScript 2016

2020
ECMAScript 2020
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

Client

Server

query
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 compile everything before **rendering** the content.
Non Single-Page Applications (SPA)

Client

HTML document

Server

request

HTML document

Download from database
Non Single-Page Applications (SPA)
Non Single-Page Applications (SPA)

Client

- HTML document
- New request
- HTML document

Server

- Database
- Data exchange
Vue.js - Single-Page Applications (SPA)

Client

Bare-bones HTML document

Server

initial request

Bare-bones HTML document

Vue.js bundle
Vue.js - Single-Page Applications (SPA)
Vue.js - Single-Page Applications (SPA)
Vue.js in nutshell
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

Server

Home | Page | About

Header component

Body component

Side component

Footer component
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.

Component structure

How we can use components?

Vue.js
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.
Methods declaration and use

Server

Vue.js

Component structure

<template>
</template>

<script>
</script>

<style>
</style>

How we can use components?

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
Vue CLI project

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

**package.json** file used by the application to know which node_modules to get.

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

**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
- store
- views
  - About.vue
  - Home.vue
- App.vue
- main.js
Vue Routers & Views
Views & Vue Router

1. Create a View

   • In the Views directory, create a new file (e.g., View1.vue).
   
   • Export the view.
Views & Vue Router

1. Create a View
   • In the Views directory, create a new file (e.g., View1.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., View1.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.
Views & Vue Router

1. Create a View
   - In the Views directory, create a new file (e.g., View1.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

Single-Page Applications (SPA)
Components

Single-Page Applications (SPA)
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.

```html
<template>
  <HTML>
  </template>
</script>
  JavaScript
</script>

<style>
  CSS
</style>

Single-Page Applications (SPA)
Components

1. Create a component
   - In components directory, create a new file (e.g., newCompo.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: "newCompo",
   }
   ```
Components

1. Create a component
   - In components directory, create a new file (e.g., newCompo.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: "newCompo", }
   ```

3. Register/Import the new component in the using 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
   <newCompo />
   <new-compo> </new-compo>
   ```
Components

1. Create a component
   - In **components** directory, create a new file (e.g., newCompo.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: "newCompo", }
   ```

3. Register/Import the new component in the using **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
   <newCompo />
   <new-compo> </new-compo>
   ```

This is a new view page

- Hi this is a new compo 1
- Hi this is a new compo 2
- Hi this is a new compo 3
- Hi this is a new compo 4
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.
Components - `<script>`

`<script>` 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 used to style an HTML document.

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

  keep an eye on the `<style>`
Data, Props and Methods
Data must be declared as a function that returns the data object(s).

**Supported data types** (String, Number, Boolean, Array, Object, Date)

**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.

**Using JavaScript Expressions:**

https://vuejs.org/guide/essentials/template-syntax.html#using-javascript-expressions
Data declaration and use

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

**Supported data types** (String, Number, Boolean, Array, Object, Date)

**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.

**Using JavaScript Expressions:**

https://vuejs.org/guide/essentials/template-syntax.html#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** can be accessed directly in the Vue instance, or they can be used 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>
```

```javascript
data: function() {
  return {
    message: "Hi, I am a v-text",
    messageHtml: "<h1> Hi, I am a v-HTML<h1/>",
  }
}
```

Hi, I am a v-text

Hi, I am a v-HTML
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>
```

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

Hello John, you have clicked 7 times
**Directives: v-show & v-once**

**v-show**: toggle the element's visibility based on the truthyness 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
<new-compo v-once msg = "This is a new compo" />
```
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.

**View: view1.vue**

```html
<como-list :compoArray="compoArray"/>
```

```javascript
data: function() {
    return {
        compoArray: [
            {id: 1, msg: 'msg description 1'}, ..
        ]
    }
}
```

**Component: compoArray.vue**

```html
<ul>
    <li v-for="compo in compoArray" :key="compo.id">
        <span>Id: {{compo.id}}</span>
        <br/>
        <span>Msg desc.: {{compo.msg}}</span>
    </li>
</ul>
```

```javascript
export default {
    name: "compoList",
    props: ["compoList"],
    data: function() {
        return {
            compoArray: [],
        }
    },
    template: 
        `<div class="item" v-for="compo in compoArray" :key="compo.id">
            <b>Id: </b>{{compo.id}}
            <br/>
            <b>Msg desc.: </b>{{compo.msg}}
        </div>
    
    }
    </script>
```
Directives: v-for

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

**View:** view1.vue

```html
<como-list :comoArray="comoArray" />
...
data: function() {
  return {
    comoArray: [
      {id: 1, msg: 'msg description 1'}, ...
    ]
  }
}
```

**Component:** comoArray.vue

```html
<ul>
  <li v-for="como in comoArray" :key="como.id">
    <span><b>Id:</b>{{como.id}}</span><br/>
    <span><b>Msg desc.:{{como.msg}}</b></span>
  </li>
</ul>
```
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()
- BeforeCreate
- Init Events & Lifecycle
- Init injections & reactivity
- Has 'el' option?
  - NO
  - Has 'template' option?
    - NO
    - Compile template into render function
    - Compile el's outerHTML as template
    - beforeDestroy
    - mounted
    - beforeUpdate
    - Virtual DOM re-render and patch
    - destroyed
    - Destroyed
  - YES
    - mounted
    - when data changes
    - beforeUpdate
    - Virtual DOM re-render and patch
    - destroyed
    - Destroyed
- beforeMounted
- mounted
- beforeUpdate
- updated
- Destroyed
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.**

**beforeDestroy**

**destroyed**

**beforeUpdate**

**updated**

**Mounted**

**Virtual DOM re-render and patch**
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.

- Has "template" option?
  - YES
    - Compile template into render function
  - NO
    - Compile el's outerHTML as template

**beforeDestroy**
Teardown watchers, child components and event listeners

**beforeUpdate**
Virtual DOM re-render and patch

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

**mounted**
when data changes

**destroyed**

**Destroyed**

**updated**
Lifecycle hooks

new Vue()

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

created

beforeMount
Create vm.$el and replace "el" with it

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.

destroyed

Destroyed

beforeUpdate

Virtual DOM re-render and patch

when data changes

Mounted

when vm.$destroy() is called

Teardown watchers, child components and event listeners

afterDestroy

Compiled template into render function *
Lifecycle hooks

**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.

**beforeCreate**
Called right before a Vue instance is created.

**created**
Called when `vm.$mount()` is called.

**beforeMount**

**mounted**
Called immediately after the element is mounted.

**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 are destroyed.
NPM
Node Package Manager
NPM

`npm`¹ is a package manager for the JavaScript programming language. `npm` has its own command-line interface (CLI). `npm` offers a registry that is an online database of public and paid-for private package. Why `npm` is very important: it allows adapting packages of code for your apps, or incorporating packages as they are.

¹https://www.npmjs.com/
Vue.js – I - summary

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. Methods declaration and use

Home | Page | About

Header component
Body component
Footer component

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

Server

Directives
A directive is a special token in the markup that tells the library to attach a specified behavior to that DOM element.
Vue.js – I - summary

Single-Page Applications (SPA)
Vue.js – I - summary

Single-Page Applications (SPA)
Vue.js - I

Try it yourself ...
Create a folder in the root directory of your project and call it “Data”, and create a file within it, `my.json`.

"Posts": is called the **top level key**, which is defined for each patch of resources.

In terminal, write:

```bash
>> npm install json-server
```

After the installation is complete, write:

```bash
>> json-server --watch Data/my.json
```

If it did not work, try:

```bash
>> npx json-server --watch
```

Resources

http://localhost:3000/Posts
• Already in the shared repo, a very simple example on using regular expressions to validate a password provided by a user.
A simple example on using Fetch API to fetch and output data in Vue.js environments.
Vue.js – Simple FAQ

WAD - frequently Asked Question (FAQ)

Do I need to attended the lecture/labs?

Attending the lecture/labs is not mandatory, but highly advised.

Do I need to physically attened the exam

Do I need to attend the discussion of my homework

I cannot attend the exam, can I retake it

There will be a resit exam offered to students who do not pass/attend ex

• A simple FAQ example on using some of the Vue.js techniques.
Usful Extensions

Install Vetur extension if you are using VSCode, or try to find a similar extension if you are not using VSCode. It will be very helpful when coding.

Devtools extension (Google Chrome/ Firefox). If installed successfully, its icon should appear among the icons of extensions in your browser.
Extra resources

• **Vue.js – Tutorial** *(https://vuejs.org/tutorial/#step-1)*: an easy step by step tutorial about the essentials of Vue.js

• **https://vuejs.org/**
Thank You for your attention

Mohamad Gharib
mohamad.gharib@ut.ee