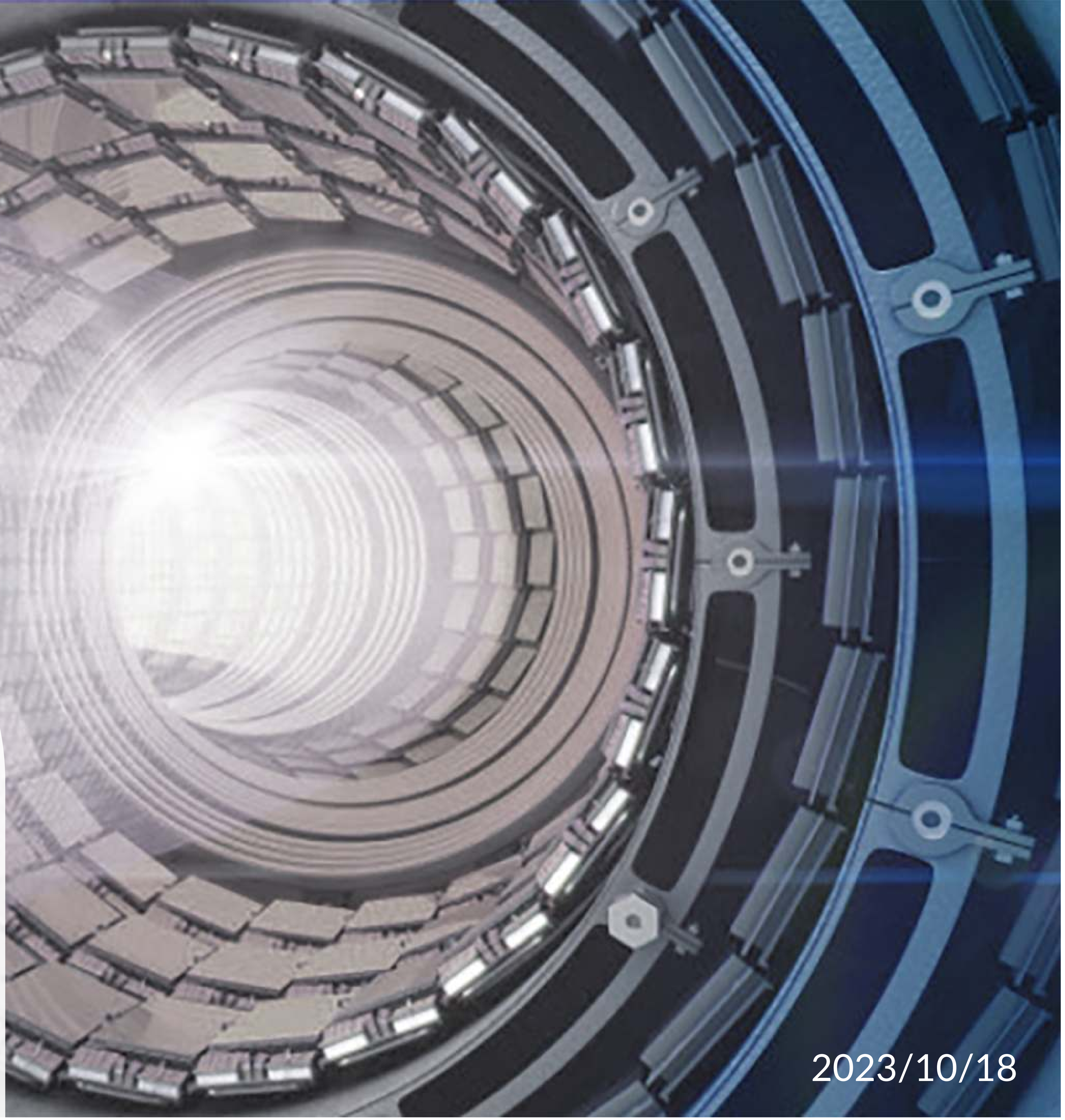


TNG TECHNOLOGY CONSULTING

Vue.js for Beginners

An introduction into Vue.js

Denise Meinhardt
Dominikus Hellgartner

2023/10/18

Preparation

- You will need an up-to-date version of npm, node and git (tested with npm 9.5.0 and node v18.15.0).
- Prepare your IDE for Vue support:
  - IntelliJ: Install vue-js plugin
  - VisualStudio Code: vue-js plugin volar
- Prepare your Browser for Vue development: Install the Vue devtools.
- Prepare the exercises:

  -
    ```
    git clone https://github.com/Hellgartner/js-days-vue-beginner-workshop.git
    ```
  - Run:

    ```
    npm ci
    npm run dev
    ```
- You should now be able to open localhost:5173 in your browser and see a white page there

Contents

Training
ExtJS
JIRA
Java
Maven
Scala
Kanban
Agile
Consulting
TNG Technology
Groovy
Cloud
TestDrivenDevelopment
Scrum
Confluence
C
WebServices
TechDay
BusinessIntelligence
Sonar
JavaScript
SQL
Coaching
InnovationGames
Python
MapReduce
BBQ
Retreat
Spring
Ruby
Team
CodeReview
Vorträge
DevOps
Datawarehouse
Grails
node.js
Eclipse
Schokolade
Perl
Hibernate
PHP
JEE
C++

What is Vue.js

„ Vue (pronounced /vjuː/, like view) is a JavaScript framework for building user interfaces. “

Source: Vue.js docs

Let's decode that a bit:

- JavaScript framework
- Main goal: Build user interfaces
- Progressive: Framework concentrates on building a *View Layer*, other functionality can be added as plugin on demand

Current versions:

- v3.3.x : Current version, improved performance and code size over v2, cool new features, few ecosystem libraries not yet migrated.
- v2.7.x : Migration release for older projects, back-ported composition API and `<script setup>` support,
  End of live at December 31st, 2023.

(This workshop uses Vue 3)

How to start a new Vue project

Vite

- With the provided create script, project setup is painless.
- Incredibly fast dev server hot reloading.
- Has access to Rollup plugins as well as to its own plugin collection.
- Also used in the official create-vue script.

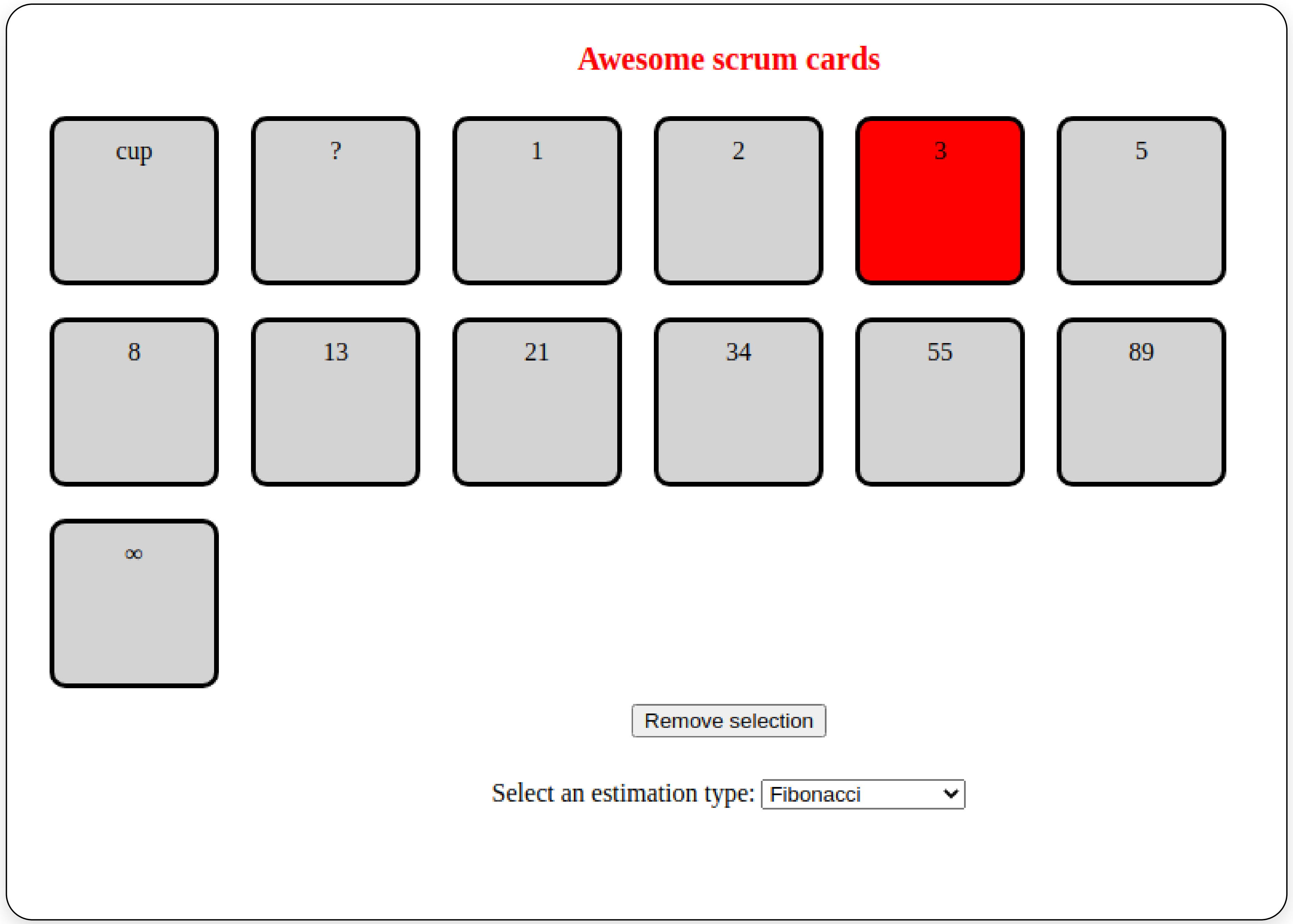
Legacy option: Vue CLI (currently in maintenance mode)

For more complex SPAs: Nuxt

- Routing, State management, Folder structure.
- Supports server side rendering and static site generation.
- Nuxt3 (compatible with Vue3) arrived at 16.11.2022

Command line TypeScript type-checking: vue-tsc

What are we going to build today



Contents

Training
ExtJS
JIRA
Java
Maven
Scala
Kanban
Agile
Consulting
TNG Technology
Groovy
Cloud
TestDrivenDevelopment
C
WebServices
Confluence
Scrum
TechDay
BusinessIntelligence
Sonar
JavaScript
SQL
Coaching
InnovationGames
BBQ
Python
Spring
CodeReview
Vorträge
MapReduce
Retreat
DevOps
Ruby
Team
Datawarehouse
Grails
Eclipse
Schokolade
node.js
Perl
Hibernate
PHP
JEE
C++

Basic structure

```
<template>
  <div class="foo"> {{ helloText }} </div>
</template>

<script setup>
    const helloText = "Hello World";
</script>

<style scoped>
    .foo { color: green; }
</style>
```

Main building blocks:

- HTML Template
- Script block
- Styling block

Script block

```
<script lang="ts" setup>
    const helloText = "Hello World";
</script>
```

Two possible options:

- Use JavaScript (see previous slide)
- Use TypeScript
- Indicate use of composition API

Every variable defined in the script block is available for use in the template (see next slide).

Template

```
<template>
  <h1>Some headline</h1>
  <div class="foo"> {{ "Say hello: " + helloText }} </div>
</template>
```

Templates can contain arbitrary HTML content

`{{ }}` allows to use the results of a javascript expression in the template
All variables defined in the script block can be used.

⚠ In Vue2, a template can have only one top-level element!

Styling (CSS) block

Possible attributes:

scoped

Makes the CSS work only on the component without leaking out.

⚠ Outer styles can still leak into the components!

lang="scss"

Use SCSS instead of CSS. The options available here depend on the installed Vite pre-processors.

Options API

For Vue 2.x + Migration

```
<script lang="ts">
import { defineComponent } from "vue";

export default defineComponent({
    name: "Component name",
    setup() {
    },
    // further options
});
</script>
```

- Component defined via one large object.
- Can be extended by using the composition API inside the setup function.
- Vue 2 only supports composition API starting with v2.7.

Exercise 1: The first component

1. Modify `ScrumEstimation.vue` to contain a headline and a div with some text below
2. Make sure you can see the component in the browser
3. Load the text in the div from a constant in the `<script>` block
4. Style the component:
    - The headline shall be red and have a font size of 20px
    - The headline as well as the content below shall be centered
5. Using the browser's CSS-inspector, investigate how the scoped styles work
6. Switch the style to SCSS
7. Locate the component using the Vue devtools

If you managed to finish early, you can try to determine how the app is mounted in the surrounding html (Hint: look at App.vue and main.ts)

Contents

Training
ExtJS
JIRA
Java
Maven
Scala
Kanban
Consulting
TNG Technology
Agile
Cloud
Groovy
TestDrivenDevelopment
Scrum
Confluence
C
WebServices
TechDay
BusinessIntelligence
Sonar
JavaScript
SQL
Coaching
InnovationGames
BBQ
Python
Spring
CodeReview
MapReduce
Retreat
Vorträge
DevOps
Ruby
Team
Datawarehouse
Grails
Eclipse
Schokolade
node.js
Perl
Hibernate
PHP
JEE
C++

Reactive data

```
<template>
  <div>{{'Hello' + text}}</div>
</template>

<script lang="ts" setup>
import { ref } from 'vue'

const text = ref<string>('World')
setTimeout(() => {text.value = 'Vue'}, 1000)
</script>
```

The **`ref`** function marks data as reactive: Usages (especially in the template) get updated on change.
In TypeScript, explicit typing is only needed if the type cannot be inferred from the initial value.
In the script block, reactive values need to be unwrapped with `.value` for reading or writing. In the template they are unwrapped automatically.

Binding attributes

```
<template>
    <a :href="linkTarget" >Click me</a>
    <a :href="linkTarget + 'fuer-bewerber/faq.html'" >Click me</a>
</template>

<script lang="ts" setup>
import { ref } from 'vue'

const linkTarget = ref<string>('https://www.tngtech.com/')
</script>
```

Binding attributes using the `:`-prefix

- binds (reactive) expressions to an attribute value
- expressions can access any variable defined in the script block

React to user actions

```
<template>
  <div>
    <div>{{ someContent }}</div>
    <button @click="resetText()">Reset text</button>
  </div>
</template>

<script lang="ts" setup>
import { ref } from 'vue'

const someContent = ref<string>('some content');

const resetText = () => {
      someContent.value = 'foo';
};
</script>
```

Any function declared in `<script lang="ts" setup>` can be called from the template.
`@event="expression"` binds the handling of an event to an expression that is called as a function.

Computed references

```
<template>
    <div>{{ concatenated }}</div>
</template>

<script lang="ts" setup>
import { ref,  computed } from 'vue'

const someContent = ref<string>('some content');
const someMoreContent = ref<string>('some more content');

const concatenated = computed<string>(() => {
    return someContent.value + ' ' + someMoreContent.value
});
</script>
```

`Computed` references:

- provide the result of the embedded function
- are cached: The function is **only** executed if a used reactive value changes
- are reactive: Updates are automatically applied to the DOM
- In TypeScript, you can enforce a type for computed properties, though type inference will usually work as well.

Exercise 2: Reactive data

1. Start from the result of the last exercise. Make the text in the `<div>` reactive.
2. Add another reactive variable and bind it to the `title` attribute of the div.
3. Select the component in the Vue devtools, change the value of the reactive variable(s) and observe the DOM changes.
4. Add a button and make the text change on button click.
5. Render the concatenation of the title and the text in the `<div>` into another `<div>`.
6. Add another button that changes the text to a different value than the first one using only one function for both.

Contents

Conditional rendering

```
<template>
    <div v-if="loading">loading</div>
    <div v-else>loaded</div>
</template>

<script lang="ts" setup>
import { ref } from 'vue'

const loading = ref<boolean>(false);
</script>
```

`v-if="condition"`, `v-else-if="condition"` and `v-else` conditionally add elements to the DOM.

There is also `v-show="condition"`, which keeps elements in the DOM but toggles the CSS `display` property.

- Prefer `v-if` for faster rendering unless you need to switch quickly off and on again or need to keep child component state/`data`

Loops

```
<template>
    <div>Shopping list</div>
    <ul>
      <li v-for="product in products" :key="product.id">
        {{ product.name }} -- {{ product.price }}
      </li>
    </ul>
</template>
```

Syntax: `v-for="singleItem in collection"`

Also a counter can be provided: `v-for="(singleItem, index) in collection"`

Can be used to iterate over entries of an object

- `v-for="value in object"`
- `v-for="(value, key) in object"`
- `v-for="(value, key, index) in object"`
- ⚠ Ordering dependent on browser

Efficient loop updates

```
<template>
    <div>Shopping list</div>
    <ul>
      <li v-for="product in products" :key="product.id">
        {{ product.name }} -- {{ product.price }}
      </li>
    </ul>
</template>
```

Vue rerenders loops if the contents of the collection change

To do so efficiently (and not mess up component state/`data`)

- add a unique `key` (`number|string|boolean`) to each element
- elements with the same `key` before/after re-rendering will use the same component instance and DOM nodes
- Other component instances and DOM nodes are destroyed/created

Input bindings

```
<template>
  <div>
    <label>Type here: <input type="text" v-model="message"/></label>
    <div>I will write what you type: {{ message }}</div>
  </div>
</template>
```

`v-model` creates a two-way data binding between the input field and a `reactive` variable

- Listens to the correct change event dependent on the input type
- Hooks the property to the correct attribute dependent on the input type

If you need to explicitly trigger side-effects on update:
Handle corresponding change events yourself

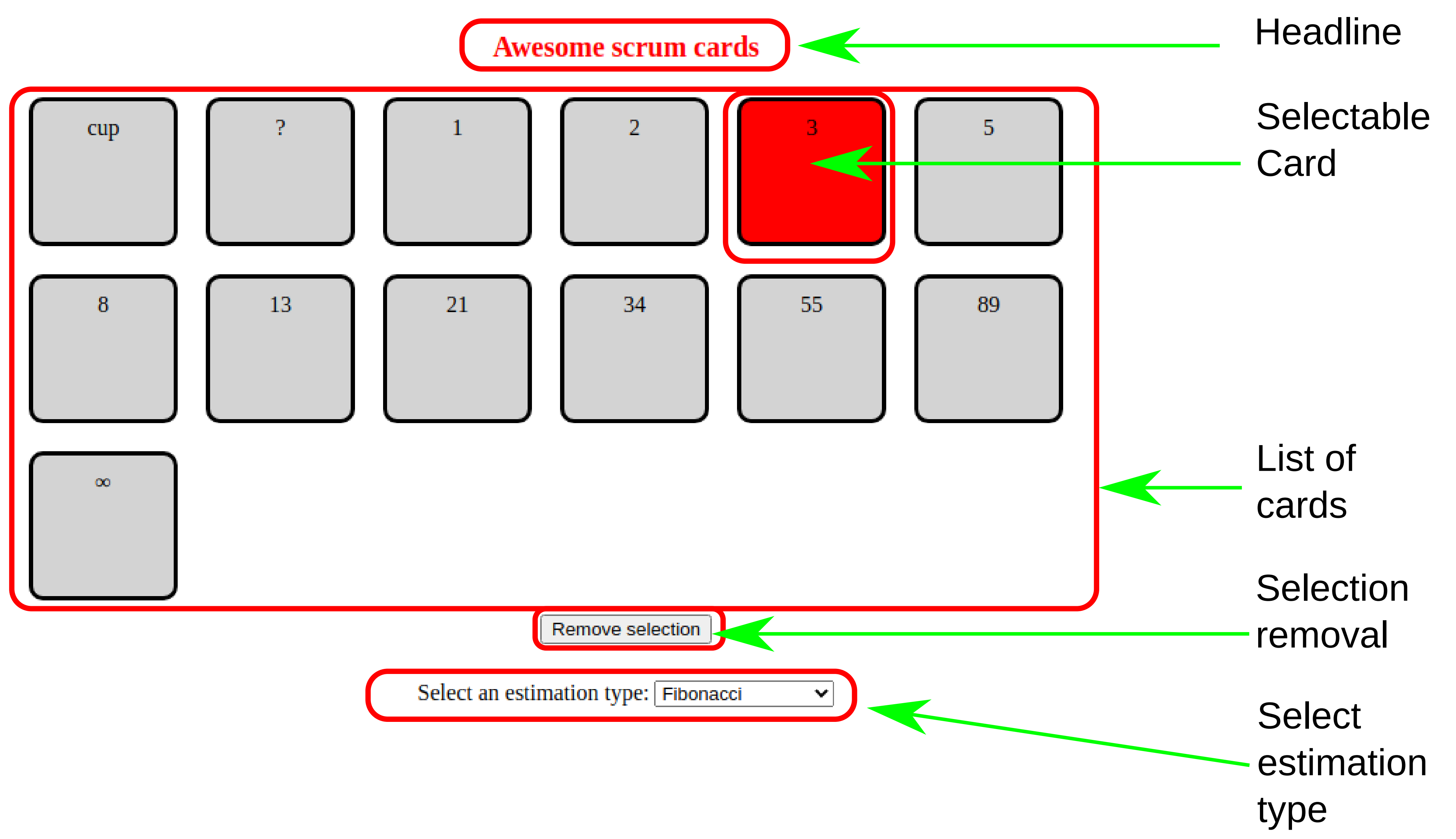
Exercise 3

1. Display all the values of the "fibonacci" scrum estimation game. You can use the predefined values:
   `import scrumEstimationValues from "@/services/scrumEstimationValuesProvider";`
2. Make it look like cards but do not spend too much time on styling.
   Hint: As this is not a CSS workshop, just import the prepared styling in the `<style>` section
   `@import "src/style/scrumestimation_card";`
   and add the `awesomescrumestimation-card` class to the cards.
3. Make the cards selectable: If a card was clicked, mark it as selected (e.g. use a red background).
4. Add a button to remove the selection.
5. Add a dropdown to switch between "Fibonacci" and "T-Shirt size" estimation
   (if you need help: v3.vuejs.org/guide/forms.html#select).

Contents

Training
ExtJS
JIRA
Java
Maven
Consulting
TNG Technology
Cloud
Agile
Groovy
Scala
Kanban
TestDrivenDevelopment
Scrum
Confluence
C
WebServices
TechDay
BusinessIntelligence
Sonar
JavaScript
SQL
Coaching
InnovationGames
BBQ
Python
MapReduce
Spring
CodeReview
Retreat
Vorträge
DevOps
Ruby
Team
Datawarehouse
Grails
Eclipse
Schokolade
node.js
Perl
Hibernate
PHP
JEE
C++

What did we achieve so far?

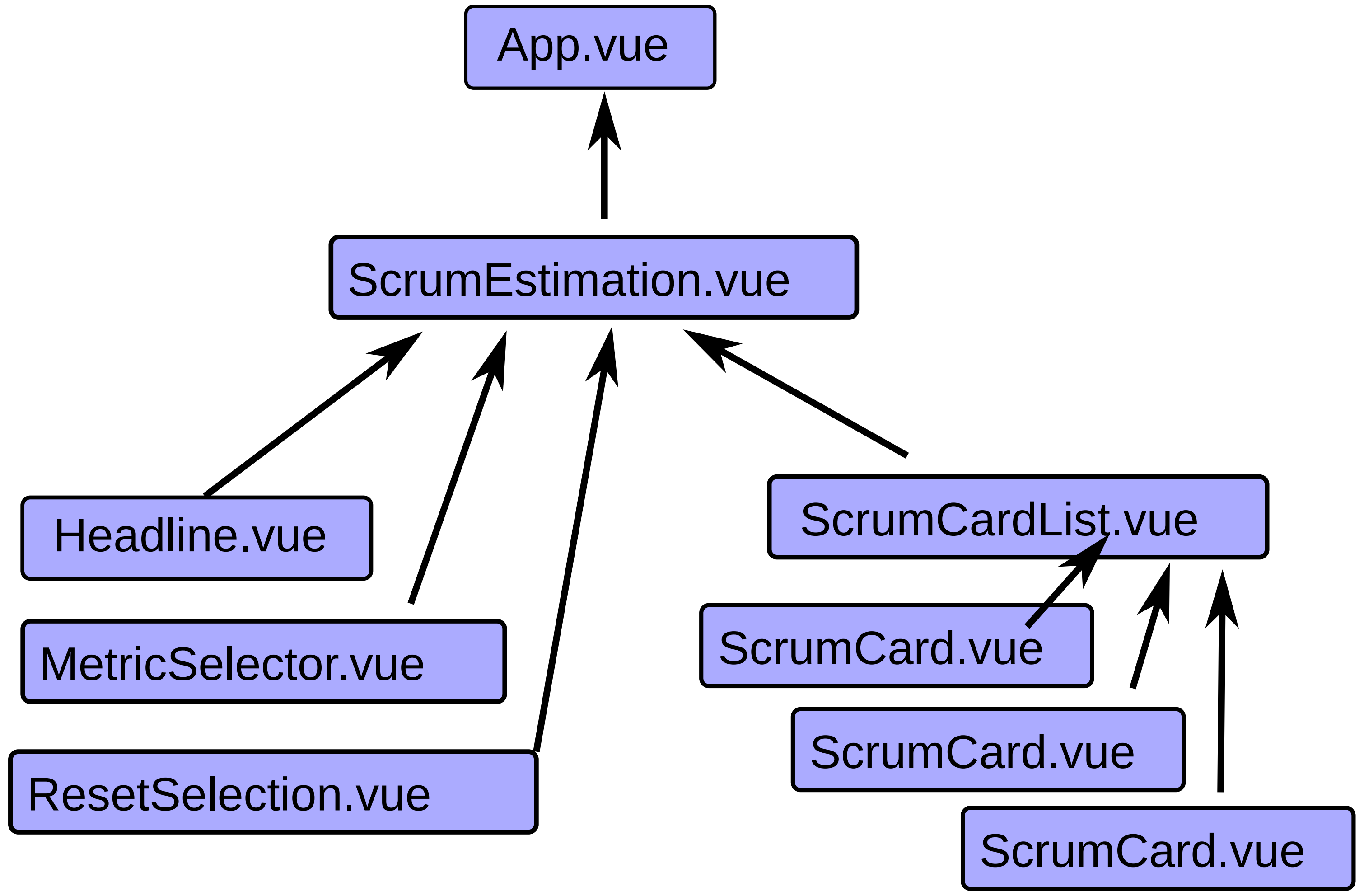


All features are in one Vue Component

- This is the Vue equivalent of the God-Class antipattern!

Sub-Components

Solution: Split into smaller components



Sub-Component usage

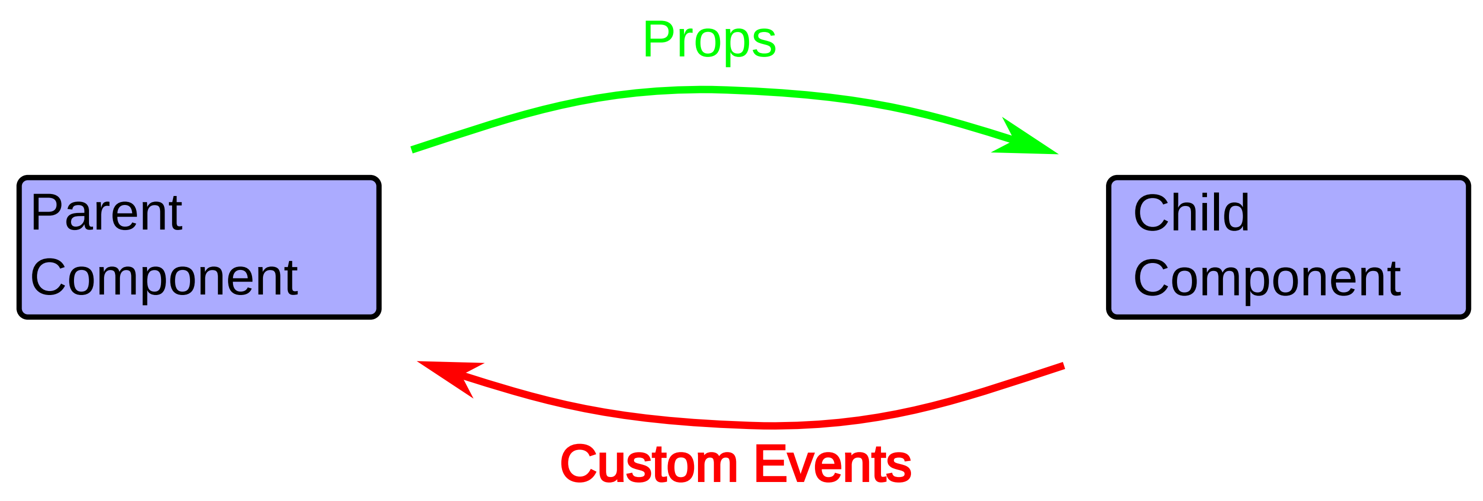
```
<template>
  <div>
      <ChildComponent/>
  </div>
</template>

<script lang="ts" setup>

import ChildComponent from "@/components/ChildComponent.vue";

</script>
```

How to use a newly created sub component?

1. Import component
2. Use component in template

Parent child communication

How to get data in and out of child components



Declaring Props

Define in child component

```
const props = defineProps({
    myStringProp: {
      type: String,
      required: false,
      default: "fooBar"
    },
    myBooleanProp: {
      type: Boolean,
      required: true
    }
  })
```

1. List the prop names as keys of the `props` object
2. Add some validation (especially type, checked at runtime)
3. Props are reactive

Declaring Props via Typescript

Define in child component

```
// without default values
const props = defineProps<{
  myStringProp?: string
  myBooleanProp: boolean
}>()

// with default values
const props = withDefaults(
  defineProps<{ prop?: string }>(),
  { prop: 'default' }
);
```

`defineProps` and `withDefaults` are compiler macros that do not need to be imported.

Passing Props

Use in parent component

```
<ChildComponent
    my-string-prop="Hello World"
    :my-boolean-prop="true"
/>
```

Props of child components can be used like normal attributes of HTML-elements

You have to use the `:`-prefix for non string props (otherwise they are interpreted as strings)

It is best practice to use kebab-case instead of camelCase for props here
(It is not strictly required for file type components but necessery if you are using "inline Vue")

Emitting custom events

```
import { defineEmits } from "vue";

const emit = defineEmits<{(e: 'my-event', value: string): void}>()

const someMethod = () => {
  emit('my-event','some value')
}
```

1. Use the `defineEmits` macro to declare your event.
2. Use the returned function to actually emit the event and to specify its payload.

Handling custom events

```
<template>
  <div>
      <ChildComponent @my-event="someMethod"/>
    </div>
</template>

<script lang="ts" setup>
import ChildComponent from 'ChildComponent';

const someMethod = (value: string) => {
  console.log(`event my-event was called with ${value} argument`);
}
</script>
```

1. Can be handled like every standard DOM event (again note the kebab-case).
2. The payload is passed to the argument of the called method

Exercise 4

1. Extract the scrum estimation card into a separate component:
   Hint: Think about the interface of the to-be-extracted component first:
   - Which properties are required?
   - Which events are required?
2. Use that component to replace the inline code in the "GodComponent"

Contents

Training
ExtJS
JIRA
Java
Maven
Scala
Kanban
Agile
Consulting
TNG Technology
Cloud
Groovy
TestDrivenDevelopment
C
WebServices
Confluence
Scrum
TechDay
BusinessIntelligence
Sonar
JavaScript
SQL
Coaching
InnovationGames
BBQ
Python
Spring
CodeReview
MapReduce
Retreat
Ruby
Team
Vorträge
DevOps
Grails
Eclipse
Datawarehouse
node.js
Schokolade
Perl
Hibernate
PHP
JEE
C++

Typical user stories

As very important manager
I want to show a grid of selectable cards to the customer showing my products
In order to ease the product selection process for the customer and thus make her/him buy more

As very important manager
I want to show a grid of selectable cards to the user showing his ToDos
In order to to allow the user to easily select his tasks to do now

"This should be a small story, you already know how to show selectable cards, right?"

Ramp up the interface

```
<template>
    <div v-if="textContent">{{textContent}}</div>
    <div v-else-if="product"><ProductComponent :product="product"/></div>
    <div v-else-if="toDoItem"><ToDoItemComponent :toDoItem="toDoItem"/></div>
</template>
<script lang="ts" setup>
import { defineProps } from "vue";

interface MyPropsInterface {
  textContent?: string
  product?: Product
  toDoItem?: ToDoItem
}

const props = defineProps<MyPropsInterface>()
</script>
```

Probably not the solution we want

Goal: Separate functionality (toggleable card) from content placed in cart

Slots: Definition

```
<template>
  <slot>
    Default content
  </slot>
</template>
```

- The position where any content can be inserted is marked with the `slot` tag
- Default content can be provided
- Multiple slots per component are possible, see named slots
- A slot can provide additional data to its content, see scoped slots

Slots: Usage

```
<template>
  <MyComponentWithSlot>
    <p>
      Whatever I want to place in the slot
    </p>
    <SomeOtherComponentIWantToUseHere/>
  </MyComponentWithSlot>
</template>
```

- The content of the tag referencing the component with the slot will be placed in the slot
- Accepts any kind of template markup
- Including use of vue components

Exercise 5

1. Extract a ToggleableCard component from the scrum estimation card. It should be a component that
    - defines a card with two states (selected and not selected) and changes its style according to the selection state.
    - gets its selection state passed from the outside.
    - has arbitrary content.
2. Use that toggleable card in the scrum estimation card.

Contents

Training
ExtJS
JIRA
Java
Maven
Scala
Kanban
Agile
Consulting
TNG Technology
Cloud
Groovy
TestDrivenDevelopment
Scrum
Confluence
C
WebServices
TechDay
BusinessIntelligence
Sonar
JavaScript
SQL
Coaching
InnovationGames
BBQ
Python
MapReduce
Spring
CodeReview
Vorträge
Retreat
DevOps
Ruby
Team
Grails
Datawarehouse
node.js
Eclipse
Schokolade
Perl
Hibernate
PHP
JEE
C++

Summary

A Vue component groups markup, functionality and styling

Using *reactive*-data, it is possible to adapt the resulting HTML as the data changes

User actions can change the data via event handling

Any (non-trivial) application should be split into multiple sub components

- Data flow from parent to child: `props`
- Data flow from child to parent: `events`

Advanced topics

- `class`/`style` bindings
- Event modifiers
- Provide/Inject
- Transitions
- Render functions
- Teleport
- Composables
- Routing (Vue Router)
- State management (Pinia)
- Testing (Vue Testing Library, Vue Test Utils)

Thank You!

Any questions?