# Multiplatform szoftverfejlesztés

## Vue

# Emlékeztető

- Mit is akarunk megoldani?
  - Automatizált HTML frissítés
    - Listákkal, feltételekkel
  - Input kezelés
  - Adatkötés (kétirányú, ha lehetséges)
  - Kompozíció
    - Felület komponensenkénti kezelése
  - Tooling
    - Debug, test

# Vue

# Hello, World

```
<template>
  <h1>Hello, {{ name }}!</h1>
</template>
<style scoped>
h1 {
  font: 48px "Segoe Print";
}
</style>
```

```
<script lang="ts">
import Vue from "vue";
export default Vue.extend({
  name: "HelloWorld",
  props: {
    name: String
  }
});
</script>
```

- .vue fájl
  - SFC: Single File Component
  - Egyben van minden: HTML, CSS, JS (TS)

# HTML template

- Saját HTML template szintaktikája van
- Deklaratív felület megadás
- {{}} létrehozza az adatkötés
- v-bind szintén adatkötés, de attribútumra
  - v-bind:title=“name”
  - Leggyakrabban használt direktíva, van rövidítése
    - :title=“name”

# Feltételes elem – v-if

- 0, vagy 1 elem generálása
- Tetszőleges kód, amire lehet hívni if-et
  - Hívhatunk függvényeket is

```
<p v-if="numbers.length>3">Sok elem</p>
```

  - Az adatnak léteznie kell a komponensben
    - data, props, …

```
  data(){ return { numbers: [1, 2, 4, 5] }; }
```

- Van v-else és v-else-if is, illetve v-show, ami csak display:none-t állít, nem törli

# Ciklus – v-for és v-key

- 0, vagy több elem generálása
- x in c szintaktika
  - c a lista
  - x az elem, amit lehet kötni

```
<p v-for="n in numbers">item {{n}}</p>
```

- Opcionálisan kulcsot adhatunk meg
  - Hogy ne generálja újra a teljes fát változásra
  - v-key=“n”
  - A kulcs probléma azonos React-ben is

# Események – v-on (@)

- HTML elem eseményeinek kezelése
- Egy függvényt adhatunk meg

```
<button v-on:click="reverseNumbers">Nyomj meg</button>
```

- Amit a methods-ban definiálunk

```
methods:{
  reverseNumbers(){
    this.numbers = this.numbers.reverse();
  }
}
```

# Események – v-on (@)

- Van pár módosító kényelmi okok miatt
  - .prevent: Meghívja a preventDefault-ot
  - .stop: stopPropagation
  - .self: csak akkor hívja meg, ha ez a vezérlő váltotta ki az eseményt
  - Stb.

```
<button @click.prevent="reverseNumbers">Gomb</button>
```

# Két irányú adatkötés – v-model

- Tipikusan input esetén
  - Ha változik az adat, akkor frissüljön az input
  - Ha a felhasználó beír valamit, akkor frissüljön az adat

```
<input v-model="price" />
```

- Tetszőleges adatra köthetünk
  - Ahogy az egyirányú esetben
  - Akár checkbox-ot string-re

# Stílus – v-bind:class (:class)

- Vesszővel elválasztott osztálylista a feltételekkel
- Objektum szintaktika
  - Az osztálynév amit rá akarok tenni
    - Nem kell ténylegesen létezzen
    - Ha van benne kötőjel
      - Idézőjel kell (‘font-size’), vagy
      - camelCase (fontSize)
  - A feltétel egy JS kifejezés, ami igaz/hamis

```
<h1 :class="{center: isCentered}">Hello, {{ name }}!</h1>
```

# Stílus – v-bind:class (:class)

- Tömb szintaktika
  - Egy tömbben felsorolva JS kifejezések
  - Mindegyiknek egy string-re kell kiértékelődni
  - Ezt az osztályt teszi rá
  - Megjegyzi, hogy melyik tette rá, így ha legközelebb mást tesz rá, akkor a régit le tudja venni

```
<h1 :class=“[centered]">Hello, {{ name }}!</h1>
```

- Automatikusan kezeli a vendor prefixeket

# Komponens

## Részei és kompozíció

# Tulajdonságok – props

- props
  - Kívülről állíthatjuk (akár adatkötéssel is)
  - HTML szerű attribútumként jelenik meg
  - Csak olvasható
  - Egyirányban köthető
  - Tömb

```
export default {
  props: ['title'],
  // data, methods, computed, components, …
}
```

# Belső állapot – data

- data
  - Belső állapot
  - Kétirányú adatkötésben is benne lehet
  - Függvény adja vissza az objektumot

```
data(){
  return {
    numbers: [1, 2, 4, 5]
  };
}
```

# Segédfüggvények – methods

- methods
  - A segédfüggvényeink helye
  - Objektumon belül függvények

```
methods:{
  reverseNumbers(){
    this.numbers = this.numbers.reverse();
  }
}
```

# Függőségi gráf – computed

- computed
  - Csak olvasható tulajdonságok
  - Függvény szintaktika (objektumon belül)

```
computed: {
  filteredNumbers() {
    return this.numbers.filter(x => x < 5);
  }
}
```

  - Függőségi gráfot épít fel
    - Akkor számolja újra, amikor valamelyik függőség változik (numbers)

# Változás figyelés – watch

- watch
  - Bármelyik adatra rátehetünk egy függvényt, ami akkor hívódik meg, amikor az adat változik
  - A függvény neve az adat neve

```
watch: {
  price(newValue: string, oldValue: string) {
    alert(oldValue + "=>" + newValue);
  }
}
```

  - Cél: debug, log, vagy átmenet kezelése
    - Például animálni a változást

# Kompozíció – components

- components
  - Gyerek komponensek felsorolása
  - Template-ben használhatók
    - Többször is, több példány jön létre

```
<script>
import Counter from './Counter.vue'

export default {
  components: {
    Counter
  }
}
</script>
<template>
  <h1>Hello, Leo!</h1>
  <Counter />
</template>
```

# Életciklus

- created: miután a komponens létrejött (konstruktor)
- mounted: benne van a virtuális fában
- updated: render után
- destroyed: megszűnt (destruktor)

- Mindegyiknek van egy before… változata
  - Az adott funkció előtt hívódik meg
  - Például beforeUpdate a render előtt

# Optimalizáció

- React-tel ellenétben itt nem kell PureComponent és társai
  - A függőségi gráf automatikusan megoldja ezt

# Tooling

- Kényelmes fejlesztéshez kell .vue fájl
  - SFC: Single File Component
    - Benne van a HTML sablon, CSS és JS/TS
  - E nélkül gyenge a keretrendszer
    - Nincs benne HTML szintaktikai elemző, stb.
- Csak olyan eszköz jöhet szóba, ami támogatja
- Szerencsére sok ilyen van
  - Pl. Visual Studio Code, Webstorm

# Kérdések?