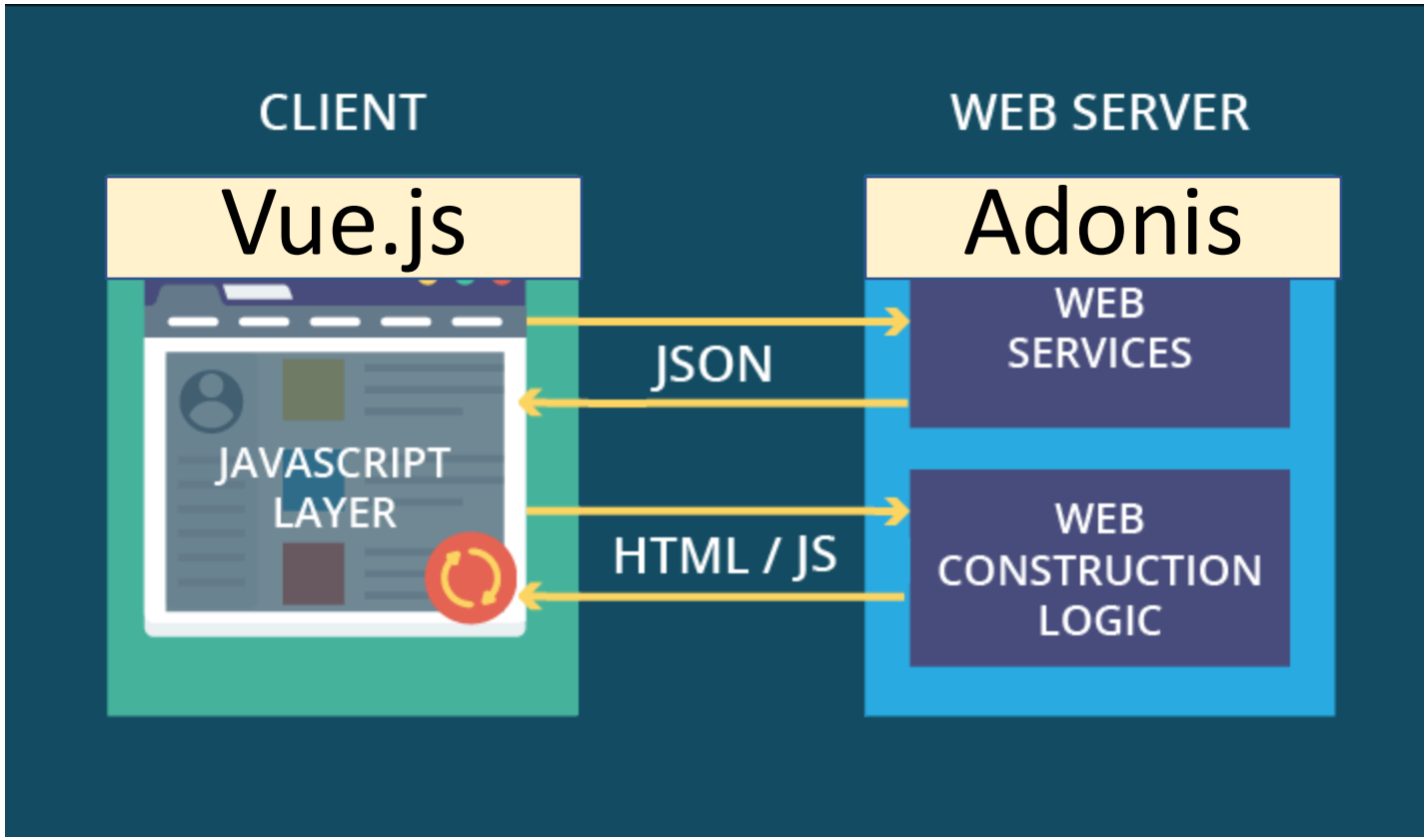
SSR, SSG, Hydration, PWA

kurz Vývoj progresívnych webových aplikácií

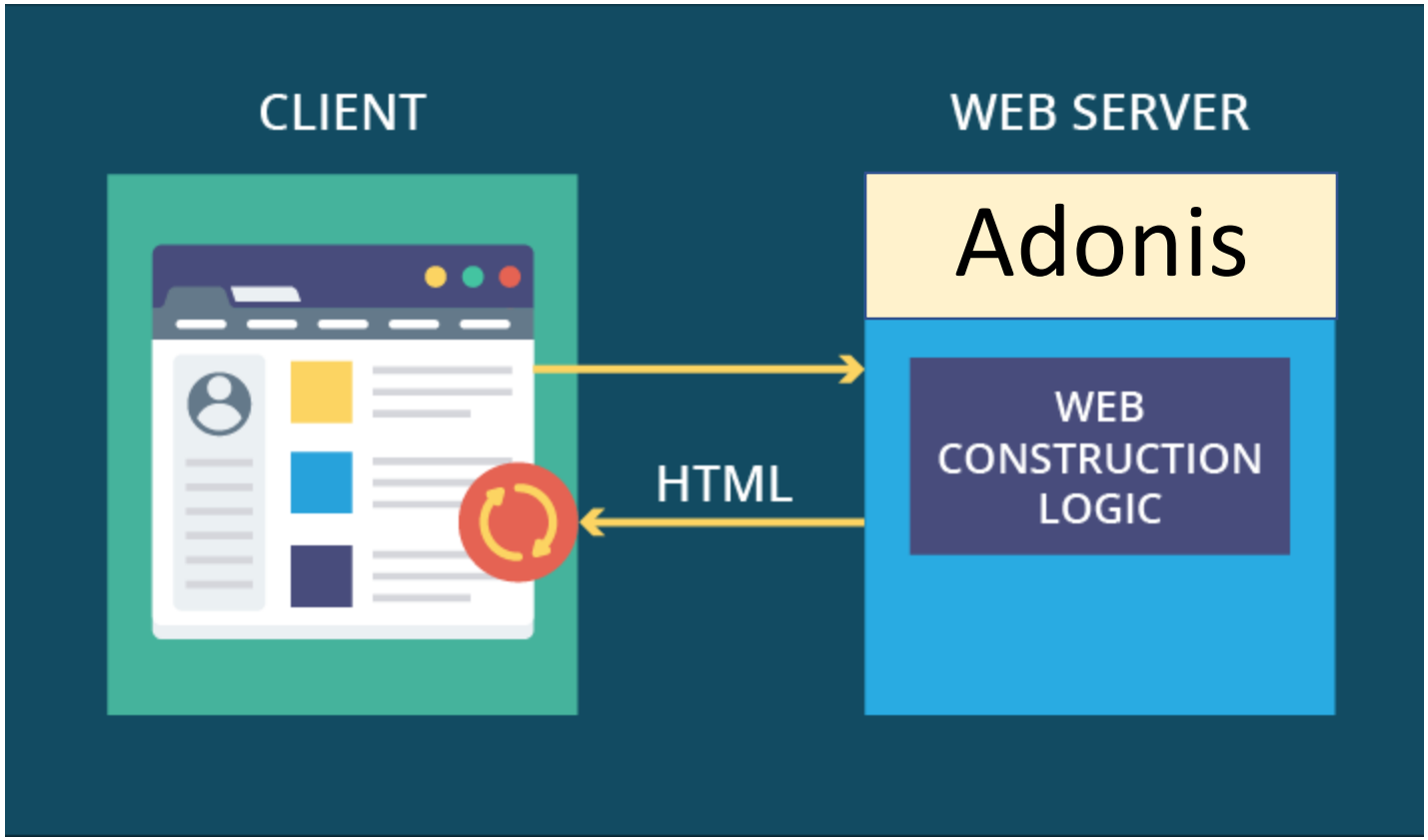
Eduard Kuric

STU FIIT

Vue.js SPA – Adonis.js



SSR Node.js (Adonis.js/Laravel)



Vieme, Vue.js ...

- Je rámec na vytváranie aplikácií na strane klienta
- Vue komponenty
  - produkujú,
  - manipulujú DOM,
  - vo webovom prehliadači.

Vue.js ako SSR

- Komponenty môžeme zostaviť na strane servera do HTML
- Vygenerované HTML vrátiť webovému prehliadaču

- **Načo Vue SSR?**
  - SPAčky si pýtajú veľa pri prvom doručení obsahu (pozrite **Core Web Vitals Metrics**)
  - Lepšie SEO optimalizácie, webové prehliadače vidia priamo zostavený HTML

Ako Vue SSR?

- Vytvorme si package.json súbor:

```
npm init -y
```

- Pridajme podporu ES modulov:

```
"type": "module"
```

- Nainštalujme vue:

```
npm install vue
```

example.js (Vue SSR)

```
import { createSSRApp } from 'vue'
import { renderToString } from 'vue/server-renderer'

const app = createSSRApp({
  data: () => ({ count: 1 }),
  template: `<button @click="count++">{{ count }}</button>`
})

renderToString(app).then((html) => {
  console.log(html)
})
```

example.js (Vue SSR)

- Spustime aplikáciu:

```
node example-vue-ssr.js

// vystup
<button>1</button>
```

Express framework

- Minimalistický webový rámec nad Node.js poskytujúci základný aparát na zjednodušenie vývoja (MVC) aplikácií
- Prináša podporu:
  - smerovanie (routing)
  - middleware
  - šablóny
  - integráciu s DB systémami
- https://expressjs.com/

- `npm install express`

Express framework (server.js)

[ukážka k prednáške]

```
node server-express.js
```

http://localhost:3000

Hydratácia (hydration)

- **hydratácia je znovunavrátenie reaktívnej podoby statickému HTML obsahu (prijatý zo serveru)**
  - prijatý (statický) HTML obsah (DOM) môžeme na klientovi **hydratovať** na dynamický obsah (reaktívny virtuálny DOM)
- Vue nájde **mapovanie medzi DOMom vráteným zo servera** a **virtuálnym domom vygenerovaným klientskou aplikáciou**
  - tomuto hovoríme hydratácia

Hydratácia (hydration) /2

`[ukážka k prednáške]`

- client-hydration.js
- server-hydration.js
- app-hydration.js – univerzálny kód, zdieľaný kód klientom/serverom

- Pozor!

https://developer.mozilla.org/en-US/docs/Web/HTML/Element/tbody#not_specifying_a_body

Static Site Generation (SSG)

- za účelom zlepšenia SEO marketingových stránok (/contact, /about) môže postačiť **pre-rendering**
- namiesto kompilácie komponentov serverom do HTML (on-the-fly), **pre-rendering vygeneruje statický HTML obsah (pri kompilácii app) pre špecifikované smerovanie (routes)**

Quasar podporuje SSR

- https://quasar.dev/quasar-cli-vite/developing-ssr/introduction
- https://quasar.dev/quasar-cli-webpack/developing-ssr/configuring-ssr
- https://quasar.dev/quasar-cli-webpack/developing-ssr/preparation

WEB API prehliadačov

- Manipulácia s dokumentmi (DOM)
- Získavanie údajov zo servera (AJAX, Fetch API)
- Vykreslenie a tvorbu grafiky (Canvas, WebGL)
- Audio a Video (HTMLMediaElement, Web Audio API, WebRTC)
- Prístup k zariadeniam (Geolocation, Notifications, Vibration API)
- Ukladanie údajov na strane klienta (Web Storage, IndexedDB)

Notification API

```
let promise = Notification.requestPermission();
var notification = new Notification("Hi there!");
```

- https://chrisdavidmills.github.io/notification-test/

Push API

- Súčasťou WEB API

Push API umožňuje webovým aplikáciám prijímať správy zo servera

- bez ohľadu na to, či je webová aplikácia v popredí
- alebo je dokonca otvorená vo webovom prehliadači (je prehliadač zapnutý)

SPA -> PWA

- Single Page Application (SPA) je vhodná na transformáciu na Progressive Web Application (PWA)
- PWA nie je konkrétny typ aplikácie, ide o **koncept, ktorý pridáva funkcionalitu offline režimu, rýchlosti a zážitku podobného natívnym aplikáciám** do webových aplikácií
- SPA poskytuje na tento účel ideálny základ
- Ak už máte SPA, transformácia na PWA je logickým krokom, ktorý môže výrazne vylepšiť jej funkčnosť a použiteľnosť

PWA – Progresive Web Apps

- Fungujú pre každého používateľa, bez ohľadu na prehliadač/zariadenie
- Vyzerajú ako natívne aplikácie, správajú sa tak, sú „súčasťou plochy“ – push notifikácie
- Vždy aktuálne (najnovšia verzia) bez potreby sťahovania celej aplikácie
- Bezpečné (HTTPS)
- Jednoducho zdielateľné cez URL
- Umožňujú pracovať offline, alebo na sieťach s nízkou prenosovou rýchlosťou

PWA – Manifest

- JSON súbor
- Obsahuje základné informácie o aplikácii:
  - Názov
  - Ikony
  - Štartovaciu URL pri spustení aplikácie
  - …

- Príklad manifestu

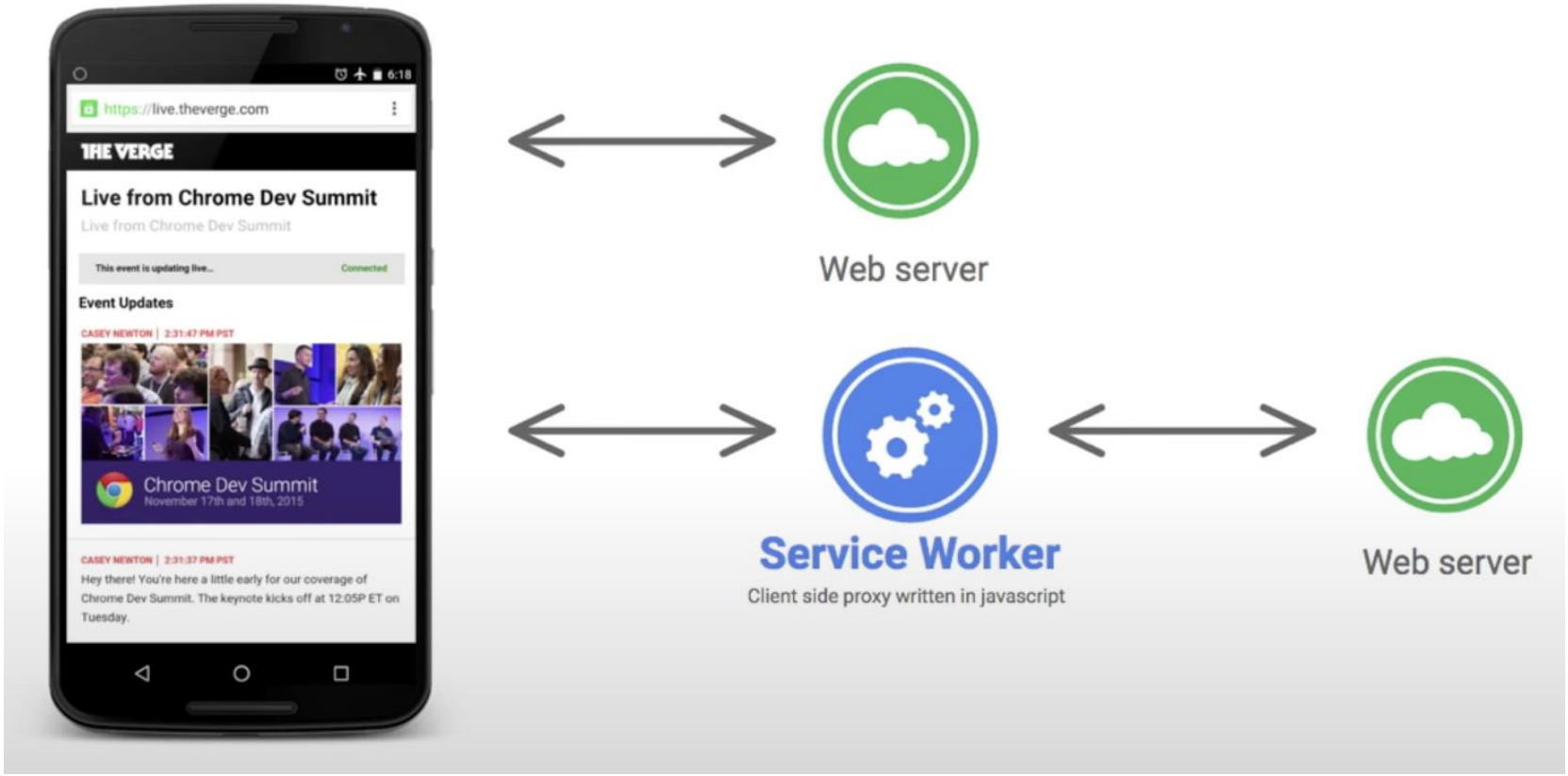
PWA – Service worker

- Service Worker pre progresívne webové aplikácie (PWA) sa nachádza vo webovom prehliadači a je registrovaný pre konkrétnu aplikáciu alebo webovú stránku
- Nie je to súčasť operačného systému, ale funguje ako súčasť prehliadača, ktorý podporuje štandardy PWA
- *DevTools → Application → Service Workers*

PWA – Service worker /2

- Service Worker API - súčasťou WEB API
- Proxy, ktoré nám umožňuje riadiť a obsluhovať požiadavky medzi aplikáciou (klientom) a web serverom
- Beží v samostatnom vlákne, nezávislé od hlavnej aplikácie
- Vyžaduje HTTPS protokol
- Scope – určuje, pre ktoré “súbory” (cesty) bude zachytávať a obsluhovať požiadavky

PWA – Service worker /2



PWA – Service worker /3

- Cacheovanie obsahu (v režime offline), napríklad obrázky
- Notifikácie/Push notifikácie (aj keď je prehliadač vypnutý)

- Introduction to Service Worker
- Using service worker
- Príklady service workerov

Quasar PWA

- `npm init quasar@latest`
- `quasar mode add pwa`
- `quasar dev -m pwa`

- `quasar build --mode pwa`

Preparation for PWA

PWA Build Commands

Sass - Syntactically Awesome Style Sheets

- CSS preprocesor
  - skriptovací jazyk, ktorý rozširuje možnosti písania CSS
  - napísaný kód sa “kompiluje” do CSS
- ďalšie známe preprocesory sú Less a Stylus

- pôvodná - indented **syntax - `.sass`**
- **nová syntax (Sass 3) Sassy CSS - `.scss`**

Sass - Prehľadnejší a organizovanejší kód

```
nav {
  ul {
    margin: 0;
    padding: 0;
    li {
      list-style: none;
    }
  }
}
```

Sass - Prehľadnejší a organizovanejší kód /2

```
CSS:
nav ul {
  margin: 0;
  padding: 0;
}
nav ul li {
  list-style: none;
}
```

Sass - Premenné

```
$primary-color: #3498db;
$font-size: 16px;

body {
  font-size: $font-size;
  color: $primary-color;
}
```

Sass - Mixiny

Mixíny sú opakovane použiteľné bloky kódu, ktoré môžu obsahovať logiku a parametre

```
@mixin border-radius($radius) {
  -webkit-border-radius: $radius;
     -moz-border-radius: $radius;
          border-radius: $radius;
}

button {
  @include border-radius(5px);
}
```

Sass - Dedenie pomocou @extend

Mixíny sú opakovane použiteľné bloky kódu, ktoré môžu obsahovať logiku a parametre

```
.button {
  padding: 10px 20px;
  border: none;
}

.primary-button {
  @extend .button;
  background-color: blue;
}
```

Sass - ďalšie...

- Sass umožňuje používať matematické operácie priamo v štýloch
- Sass podporuje cykly a podmienky, čo umožňuje dynamické generovanie kódu

WEB Components

- Web Components sú moderná technológia v HTML, ktorá umožňuje vytvárať opakovane použiteľné, zapuzdrené komponenty s vlastnou štruktúrou, logikou a štýlmi
- Sú štandardizovanou súčasťou webu, podporovanou väčšinou moderných prehliadačov, a pozostávajú z 3 hlavných častí

WEB Components /2

- **Custom Elements (Vlastné prvky):**
  - Umožňujú vytvárať vlastné HTML tagy s vlastnou logikou
  - Vytvárajú sa registráciou vlastného elementu pomocou API `customElements.define`
- **Shadow DOM:**
  - Poskytuje izoláciu štruktúry a štýlov komponentu, aby nedochádzalo ku konfliktom so zvyškom stránky.
  - Umožňuje vytvárať skryté časti DOM.
- **HTML Templates a Slots:**
  - Templates: Definujú opakovane použiteľné HTML štruktúry, ktoré sa vykresľujú iba po ich inicializácii
  - Slots: Umožňujú vkladať obsah z rodičovského DOM do Web Component

WEB Components /3

- **Kľúčové časti Shadow DOM**
  - **Shadow Host:** Prvok, ktorý obsahuje shadow tree. Napríklad `<custom-element>` je hostiteľ
  - **Shadow Root:** Koreňová časť shadow tree, kde začína obsah shadow DOM
  - **Shadow Tree:** Samotná štruktúra vnútorných prvkov a štýlov, ktoré sú zapuzdrené

WEB Components /4

- **HTML:**

```
<custom-element></custom-element>
```

- **JavaScript na vytvorenie Shadow DOM:**

```
const customElement = document.querySelector('custom-element');
const shadowRoot = customElement.attachShadow({ mode: 'open' });
shadowRoot.innerHTML = `
  <style>
    p {
      color: red;
    }
  </style>
  <p>Text vo vnútri Shadow DOM</p>
`;
```

WEB Components /4

- **Výsledok v DOM:**

```
<custom-element>
  #shadow-root (open)
    <style>
      p {
        color: red;
      }
    </style>
    <p>Text vo vnútri Shadow DOM</p>
</custom-element>
```

https://developer.mozilla.org/en-US/docs/Web/API/Web_components

Shadow DOM

- **Shadow DOM** je technológia poskytovaná štandardom **Web Components**
- **https://developer.mozilla.org/en-US/docs/Web/API/Web_components/Using_shadow_DOM**
- **U**možňuje vytvárať izolované časti DOM (Document Object Model) pre konkrétne komponenty
- Shadow DOM zapúzdruje obsah a štýly komponentu, čím zabraňuje konfliktom s inými časťami aplikácie a vytvára lepšie oddelenie logiky, štýlov a štruktúry

Shadow DOM /2

- Shadow DOM vytvára tzv. **shadow tree** – podstrom DOM, ktorý je oddelený od hlavného dokumentového stromu. To znamená, že:
- **Štýly:** Štýly aplikované na shadow tree neovplyvnia hlavný DOM (a naopak).
- **Selektory:** Bežné CSS selektory nemôžu priamo pristupovať k prvkom v shadow tree.
- **Logika:** Obsah shadow tree je viditeľný len cez špecifické API, čo zabezpečuje izoláciu.

Sass - Syntactically Awesome Style Sheets

- Komponenty vo frameworkoch ako **Vue.js**, **Angular**, **React (so Shadow DOM)** alebo **Web Components** často zapuzdrujú svoje prvky pomocou Shadow DOM
- Ich interné štýly sú izolované a externé štýly ich nemôžu ovplyvniť. **Deep selektor** umožňuje “preniknúť” cez túto bariéru

Sass - Syntactically Awesome Style Sheets

```
<style lang="scss" scoped>
.q-btn {
  :deep(.q-btn__content) {
    color: turquoise;
  }
}
</style>
```