Katedra informatiky
Přírodovědecká fakulta
Univerzita Palackého v Olomouci

DIPLOMOVÁ PRÁCE

Analýza a vylepšení webové aplikace

2024

Bc. Milan Vojáček

Vedoucí práce:
doc. RNDr. Tomáš Masopust, Ph.D., DSc.

Studijní program: Informatika, prezenční forma

Bibliografické údaje

Autor:	Bc. Milan Vojáček
Název práce:	Analýza a vylepšení webové aplikace
Typ práce:	diplomová práce
Pracoviště:	Katedra informatiky, Přírodovědecká fakulta, Univerzita Palackého v Olomouci
Rok obhajoby:	2024
Studijní program:	Informatika, prezenční forma
Vedoucí práce:	doc. RNDr. Tomáš Masopust, Ph.D., DSc.
Počet stran:	48
Přílohy:	elektronická data v úložišti katedry informatiky
Jazyk práce:	český

Bibliographic info

Author:	Bc. Milan Vojáček
Title:	Analysis and improvement of web application
Thesis type:	master thesis
Department:	Department of Computer Science, Faculty of Science, Palacký University Olomouc
Year of defense:	2024
Study program:	Computer Science, full-time form
Supervisor:	doc. RNDr. Tomáš Masopust, Ph.D., DSc.
Page count:	48
Supplements:	electronic data in the storage of department of computer science
Thesis language:	Czech

Anotace

Tato práce bude zkoumat efektivitu webového přehrávače, který je použit v nástroji Smartlook. V první části práce bude představeno webové nahrávání, jeho základní koncept. Následující část bude obsahovat teoretické aspekty práce, a popis použitých technologií. Další část bude zaměřena na rozebrání stavu přehrávače před vylepšeními a identifikace možných vylepšení. V závěrečné části bude prezentována implementace vybraných vylepšení, včetně technického popisu provedených změn a jejich očekávaného dopadu na celkovou efektivitu webové aplikace. Cílem této práce bude poskytnout pohled na úskalí při implementaci webových aplikací a demonstrovat praktické způsoby, jak mohou být řešeny.

Synopsis

This thesis will investigate the effectiveness of the web player that is used in the Smartlook tool. In the first part of the work, basic concept of web recording will be presented. The following part will contain the theoretical aspects of the work and a description of the technologies used. The next part will focus on breaking down the state of the player before the improvements and identifying possible improvements. In the final part, the implementation of selected improvements will be presented, including a technical description of the changes made and their expected impact on the overall efficiency of the web application. The aim of this thesis will be to provide an insight into the pitfalls in implementing web applications and demonstrate practical ways in which they can be solved.

Klíčová slova: Web; JavaScript; event loop; optimalizace výkonu; minimalizace spotřeby zdrojů; virtuální DOM

Keywords: Web; JavaScript; event loop; performance optimization; minimizing resource consumption; virtual DOM

Děkuji svojí rodině za trpělivou podporu při vytváření této práce.

Odevzdáním tohoto textu jeho autor/ka místopřísežně prohlašuje, že celou práci včetně příloh vypracoval/a samostatně a za použití pouze zdrojů citovaných v textu práce a uvedených v seznamu literatury.

Obsah

Seznam obrázků

Seznam zdrojových kódů

1 Úvod

Webové aplikace jsou nedílnou součástí všech mobilních zařízení a počítačů. Z technických aspektů aplikací je nejznatelnější plynulost uživatelského rozhraní. Zachovat plynulost aplikace a zároveň účinně využívat výpočetní prostředky je nezbytný požadavek všech aplikací. V této práci budou představeny některé možnosti, jak těchto aspektů docílit.

Tato práce se zabývá analýzou a vylepšením webové aplikace. Zkoumanou aplikací je přehrávač webových nahrávek, který je integrován v aplikaci Smartlook. Záměrem práce je nalézt metody, jak přehrávač vylepšit z pohledu uživatele a také z pohledu vývojáře.

Tento přehrávač byl vybrán proto, že se v něm skrývá potenciál kvůli některým naivním přístupům. Zároveň má přehrávač v některých extrémních případech výkonostní nedostatky dané zpracováním velkého množství dat. Aplikace bude pomocí dostupných nástrojů prozkoumána a budou identifikovány potenciální místa pro zlepšení. Tyto nedostatky budou demonstrovány na příkladech přiložených k práci.

Téma je relevantní pro webovou platformu, neboť s podobnými problémy se potýká téměř každá netriviální aplikace. Představená vylepšení jsou využitelná i v jiných aplikacích na webu.

2 Webové nahrávky a jejich vlastnosti

Webové nahrávky vznikají za účelem analýzy aplikací. Pomáhají identifikovat chyby, vyhledávat nejasnosti v uživatelském rozhraní, nebo měřit efektivitu marketingových kampaní. Tyto nahrávky pak slouží jako vstupní data pro analytické platformy. Nahrávky obsahují strukturu webu přesně tak, jak ji prohlížeč zobrazoval uživateli. Není to tedy jen vizuální záznam, ale přesná rekonstrukce obsahu, který byl zaznamenán při nahrávání.

2.1 Nahrávání

Nahrávání probíhá prostřednictvím skriptu přidaného do sledované aplikace, tento skript v sobě obsahuje identifikátor projektu, do kterého se data nahrávají. Při nahrávání se monitoruje chování aplikace – změny v objektovém modelu a kaskádových stylech. Využívá se MutationObserver a Monkey patching prototypů a metod. Chování uživatele je monitorováno pomocí událostí.

Všechny tyto informace jsou serializovány s vynecháním citlivých dat a persistovány v úložišti, kde zůstávají několik týdnů nebo měsíců, během kterých z nich mohou být generovány výstupy. Některé nahrané informace se také používají k další analýze, například pro tepelné mapy nebo konverzní cesty (funnely).

2.2 Přehrávání

Při přehrávání jsou nahrané informace použity k rekonstrukci relace, vytvoří se nahraný objektový model a nad ním se aplikují všechny typy událostí, jako interakce, vyplňování vstupů a podobně. Přehrávaný obsah připomíná video, ve skutečnosti se ale jedná o duplikovaný obsah nahrávané stránky. Přehrávač má za cíl replikovat nahraný obsah co možná nejpřesněji, pouze s nutnými změnami týkající se citlivých dat nebo technickým odlišnostem daným odlišným prostředím při přehrávání.

2.3 Zdroje webových nahrávek

Webové nahrávky nejčastěji vznikají pomocí analytických nástrojů pro sledování chování uživatelů. Trh s těmito nástroji je rozvinutý a obsahuje mimo Smartlook také nástroje jako Hotjar, Mouseflow, FullStory a spoustu dalších. Všechny tyto nástroje poskytují, alespoň na první pohled, podobnou funkcionalitu. Přidávají nad webové nahrávky určitou možnost analýzy uživatelských akcí a umožňují identifikovat vzorce jakými uživatelé s aplikacemi interagují. Lze pomocí nich zjistit, kde přesně uživatelé opouští stránku nebo kde narážejí na problémy. [1]

3 Web

V této kapitole jsou popsány základní pojmy týkající se tématu webových stránek. V následujících odstavcích jsou tak nastíněny některé aspekty webových prohlížečů, fungování event loop a aplikační rozhraní pro práci s ním. Dále pak důležitý prvek iframe, virtuální DOM a vývojářské nástroje v prohlížečích.

3.1 JavaScript

„JavaScript je programovací jazyk webu. Naprostá většina webových stránek používá JavaScript a všechny moderní webové prohlížeče – na stolních počítačích, tabletech a telefonech – obsahují interprety JavaScriptu, díky čemuž je JavaScript nejrozšířenějším programovacím jazykem v historii. V posledním desetiletí Node.js umožnil programování JavaScriptu mimo webové prohlížeče a dramatický úspěch Node.js znamená, že JavaScript je nyní také nejpoužívanějším programovacím jazykem mezi vývojáři softwaru.“ [2]

JavaScript se poprvé objevil v roce 1995 s primárním cílem validovat vstupy formulářů. Validace vstupu byla předtím řešena až na serveru, což znamenalo při vyplňování zbytečné volání s nevalidními daty. [3]

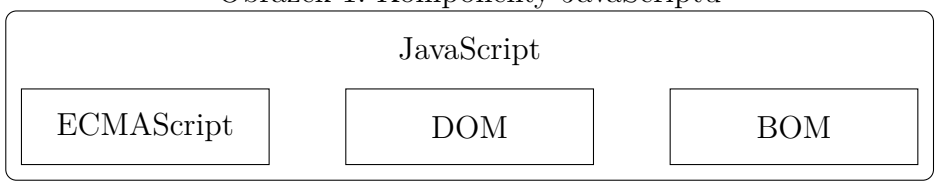
Je důležité poznamenat, že JavaScript nemá mnoho společného s programovacím jazykem Java. Java v roce 1995 nabírala na popularitě a název JavaScript toho měl využít. [4]

Od validace formulářových vstupů se JavaScript vyvinul k tvorbě plnohodnotných aplikací.

3.1.1 Implementace JavaScriptu

JavaScript a ECMAScript jsou často používána jako synonyma, JavaScript toho ale obsahuje podstatně víc, než je definováno ve standardu ECMA-262. Implementace JavaScriptu se skládá ze tří částí, jádra založeném na EcmaScript specifikaci, objektovém modelu dokumentu a objektovém modelu prohlížeče. [3] [5]

Obrázek 1: Komponenty JavaScriptu



3.1.1.1 ECMAScript

ECMA-262 definuje ECMAScript jako základ, na kterém lze stavět robustnější skriptovací jazyky. Mezi implementace ECMAScript patří webové prohlížeče, serverová platforma Node.js nebo zastaralý Adobe Flash. ECMAScript popisuje aspekty jazyka. Mezi ně patří následující. [3]

- Syntaxe (Syntax),
- Typy (Types),
- Příkazy (Statements),
- Klíčová slova (Keywords),
- Rezervovaná slova (Reserved words),
- Operátory (Operators),
- Globální objekty (Global objects)

ECMAScript prošel překotným vývojem, v roce 2015 přišla verze ES6 (také ES2015, ES Harmony) přinášející podporu spousty nových prvků jazyka. Od tohoto roku vychází vždy v červnu verze pro daný rok, tedy ES2015, ES2016 a tak dále.

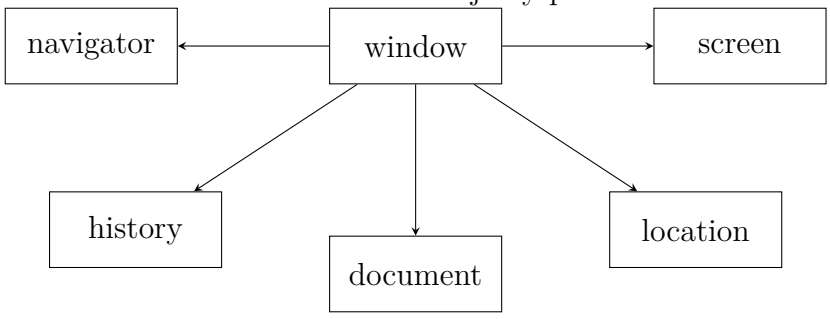
3.1.1.2 Objektový model dokumentu (DOM)

Objektový model dokumentu (Document Object Model) je aplikační rozhraní (API) pro XML rozšířené pro použití HTML. DOM udržuje celou stránku jako strukturu uzlů. Každá část stránky je uzlem obsahujícím různé druhy dat. [3] Prostřednictvím DOM API je možné plně kontrolovat stromovou reprezentaci dokumentu. Uzly je možné jednoduše přidávat, mazat, nahrazovat nebo upravovat. [3]

3.1.1.3 Objektový model prohlížeče (BOM)

Obsluha prohlížeče začíná přístupem do `window` objektu. Toto API se nazývá Objektový model prohlížeče (Browser Object Model). [6] Zajímavostí (a někdy i komplikací) je, že pro BOM, narozdíl od ostatních částí JavaScriptové implementace, dlouho neexistoval standard. Proto jej prohlížeče implementovaly po svém až do příchodu HTML5. HTML5 přineslo velkou část BOM do formální specifikace. [3]

Obrázek 2: Primární objekty prohlížeče



3.1.2 Iterátory a generátory

Iterátory v JavaScriptu jsou objekty, které umožňují vývojářům procházet kolekce, jako jsou pole nebo řetězce. Iterátory poskytují mechanismus pro přístup k prvkům v kolekci jeden po druhém, aniž by bylo nutné znát podrobnosti o struktuře samotné kolekce.

Iterátor v JavaScriptu má metodu `next()`, která při každém zavolání vrátí další prvek v kolekci. Pokud nejsou k dispozici žádné další prvky, vrátí objekt, jehož vlastnost `done` je nastavena na `true`. Iterátory jsou základem pro některé klíčové JavaScriptové funkce, jako jsou cykly `for...of`, operátor spread (`...`) a destrukturalizace.

Generátor je speciální typ funkce v JavaScriptu, který může vrátit (yield) více než jednu hodnotu, a to postupně, ne všechny najednou. Toho je dosaženo pomocí klíčového slova `yield`. Generátory v JavaScriptu jsou užitečné pro práci se sekvencemi dat, které nejsou všechny dostupné najednou, například pro čtení velkých souborů, přenos dat po síti a podobně. Generátorové funkce jsou definovány pomocí syntaxe `function*` a vracejí Iterátor. Tento objekt má metodu `next()`, která vrátí následující hodnotu generovanou funkcí.

```
function* simpleGenerator() {
  yield 1;
  yield 2;
  yield 3;
}

const iterator = simpleGenerator();

console.log(iterator.next().value); // 1
console.log(iterator.next().value); // 2
console.log(iterator.next().value); // 3
console.log(iterator.next().value); // undefined

// iterátor lze použít také ve for .. of cyklu
for (const value of simpleGenerator()) {
  console.log(value);
}
```

Zdrojový kód 1: Jednoduchý příklad použití generátoru

3.1.3 MutationObserver

MutationObserver je rozhraní webového API, které poskytuje možnost sledovat změny v DOM stromu, jako je přidávání, odstranění nebo modifikace dětských uzlů. MutationObserver je vytvořen pomocí konstruktoru MutationObserver, který jako argument přijímá zpětné volání, které je vyvoláno v případě jakékoliv změny sledovaného DOM. [7]

```
const observer = new MutationObserver(callback);
```

```
2 observer.observe(targetNode, config);
```

Zdrojový kód 2: Příklad použití MutationObserver

V příkladu 3.1.3 `callback` je funkce, která se vyvolá při každé pozorované mutaci. `targetNode` je uzel DOM, který se má sledovat, a config je objekt, který definuje, jaké typy mutací se mají sledovat (např. pouze změny atributů, nebo i změny potomků, a podobně).

MutationObserver je užitečný pro práci s dynamickými webovými aplikacemi, kde se DOM může často měnit. Může být použit například pro sledování změn v uživatelském rozhraní, detekování změn provedených jinými skripty na stránce, nebo pro implementaci reaktivních programovacích modelů.

Události v JavaScriptu

Události v JavaScriptu jsou akce, které se dějí v prohlížeči, na které může reagovat kód. Tyto akce mohou být iniciované uživatelem, jako kliknutí myší, stisk klávesy, pohyb myší, dotyk na obrazovce, atd., nebo mohou být generovány prohlížečem, jako načtení stránky, změna velikosti okna, změna obsahu formuláře, atd. [8]

3.1.4 Základní koncepty a syntaxe práce s událostmi v JavaScriptu

Aby bylo možné reagovat na událost, je třeba nejprve přidat posluchač události na objekt, který tuto událost generuje. To se dělá pomocí metody *addEventListener*, která přijímá jako argumenty typ události a funkci, která se má zavolat, když se událost vyskytne. Kód 3.1.4 demonstruje získání pozice myši v události kliknutí.

```
1 document.querySelector("button").addEventListener("click", function
     (event) {
2   console.log(
3     `Klik na tlačítko na souřadnicích: ${event.screenX}, ${event.
        screenY}`
4   );
5 });
```

Zdrojový kód 3: Událost – výpis pozice myši v okamžiku kliknutí

Objekt události Když se událost vyskytne, prohlížeč vytvoří objekt události, který obsahuje informace o této události, a předá ho jako argument funkci zavolané posluchačem události. Tento objekt může obsahovat různé informace, v závislosti na typu události, jako například pozici myši, stisknutou klávesu, a podobně.

Bublání a zachycení Když se událost vyskytne na určitém elementu, posluchače na tomto elementu a na všech jeho rodičích se zavolají v určitém pořadí,

známém jako fáze bublání a zachycení. To je důležité pro správnou práci s událostmi, které mohou být zpracovány více než jedním posluchačem.

3.2 Výpočet v prohlížeči

3.2.1 Hlavní vlákno

JavaScript je jednovláknový jazyk, což znamená, že může zpracovávat v daném okamžiku jednu operaci a musí tuto operaci dokončit, než přejde k další. Toto je často označováno jako *hlavní vlákno* (*main thread*). Hlavní vlákno zpracovává kód sekvenčně, to znamená, že vykonává v jeden čas pouze jednu operaci

3.2.1.1 Blokující a neblokující operace

Pokud je operace blokující (například čtení z disku, zasílání HTTP požadavku, atd.), hlavní vlákno musí čekat, dokud tato operace neskončí. To může vést k tomu, že aplikace „zamrzne“ a nebude reagovat na uživatelské akce. Aby se tomu předešlo, JavaScript podporuje neblokující operace pomocí asynchronních callbacků (zpětných volání), Promise [9] nebo async/await.

3.2.1.2 Více vláken

JavaScript v prohlížeči umožňuje použít více vláken pomocí Web Workers. [10] Web Workers umožňují vytvářet oddělená vlákna v prohlížeči. Každý Worker běží v samostatném vlákně a komunikuje s hlavním vláknem prostřednictvím posílání zpráv. Web workers mají omezené možnosti, nelze v nich přistupovat k `window` objektu.

3.2.2 Event loop

Event loop je základním konceptem v některých programovacích jazycích a systémech, které využívají asynchronní programování, jako je například JavaScript v prostředí Node.js nebo prohlížečích. Event loop umožňuje neblokující provádění kódu, asynchronní volání a efektivní zpracování událostí. Když hlavní vlákno narazí na asynchronní operaci, předá ji do event loopu a pokračuje dál. Jakmile je asynchronní operace dokončena, její callback se přidá do fronty událostí. Event loop poté pravidelně kontroluje hlavní vlákno, zda je volné, a pokud ano, přesune callback do hlavního vlákna k vykonání.

3.2.2.1 Princip práce

Event loop běží v nekonečném cyklu, kde sleduje a spravuje události a volání zpětných funkcí (callbacků). Jeho hlavním úkolem je sledovat, zda jsou nějaké úkoly k provádění v různých frontách událostí (například zpětných volání, mikrotasků).

Asynchronní operace V prostředích, jako je JavaScript, většina I/O operací (např. práce se soubory, síťové požadavky) je asynchronní. To znamená, že kód může pokračovat dále bez čekání na dokončení těchto operací. Event loop zajišťuje, že jakmile je asynchronní operace dokončena (např. data jsou přijata ze serveru), odpovídající zpětná volání jsou zařazena do fronty k pozdějšímu zpracování.

Fronty událostí Event loop pracuje se dvěma typy front:

- Mikrotasky – Pro operace jako jsou Promise.,
- Makrotasky – Pro většinu asynchronních operací, jako jsou setTimeout, I/O operace atd.

Cyklus zpracování Event loop neustále prochází frontami a zpracovává zpětná volání. Začíná s mikrotasky, které mají prioritu a provádí se ihned po dokončení aktuálního skriptu. Poté se přesunuje k makrotaskům.

Význam pro výkon a UX Díky event loopu může aplikace zůstat reaktivní a efektivně zpracovávat uživatelské vstupy, síťové požadavky a další asynchronní operace. To umožňuje vytvářet příjemné uživatelské rozhraní a efektivně pracovat s I/O operacemi bez zamrzání aplikace.

Event loop je klíčový pro asynchronní chování v JavaScriptu a jeho porozumění je důležité pro každého, kdo pracuje s JavaScriptem nebo jinými jazyky a prostředími, které využívají podobný model asynchronního programování. [11]

3.2.3 Fronty událostí v event loop

V JavaScriptu existují dvě hlavní fronty pro správu událostí: makroúkolová a mikroúkolová. Obě fronty jsou důležité pro správu událostí a asynchronního chování v JavaScriptu. Když JavaScriptový runtime dokončí synchronní kód, podívá se na makroúkoly. Pokud jsou nějaké k dispozici, vezme první a spustí jej.

Nicméně, pokud existuje jakýkoliv mikroúkol, JavaScript se pokusí nejprve zpracovat všechny mikroúkoly a teprve potom se vrátí k makroúlům. To je důležité pro řízení pořadí, ve kterém se asynchronní operace vykonávají. Toto je mechanismus, který dovoluje JavaScriptu zpracovávat události asynchronně tím, že je zařadí do fronty a zpracuje je, když je hlavní vlákno volné. [12] [13] [14]

3.2.3.1 Makroúkolová fronta

Makroúkolová fronta, někdy také nazývaná jako task queue nebo event queue, je jedním z typů front v JavaScriptu, který spravuje takzvané „makroúkoly“. Makroúkoly jsou většinou tvořeny událostmi jako setTimeout, setInterval, setImmediate, AJAX požadavky, a UI rendering. Když hlavní vlákno JavaScriptu dokončí veškerý synchronní kód, podívá se na makroúkoly. Pokud jsou nějaké

k dispozici, vezme první z a spustí ho. Jakmile je tento úkol dokončen, JavaScript se podívá do fronty znovu a vezme další úkol k zpracování, a tak dále. Nicméně, pokud existují nějaké mikroúkoly, JavaScript se pokusí zpracovat všechny tyto úkoly předtím, než se vrátí k makroúkolům. To znamená, že mikroúkoly mají vyšší prioritu než makroúkoly. Toto uspořádání umožňuje JavaScriptu efektivně řídit asynchronní operace a udržovat plynulost uživatelského rozhraní.

3.2.3.2 Mikroúkolová fronta

Mikroúkolová fronta spravuje takzvané „mikroúkoly“. Mikroúkoly jsou malé jednotky práce, které jsou naplánovány k vykonání po dokončení aktuálně běžícího skriptu, ale předtím, než prohlížeč provede jakékoliv další události, makroúkoly nebo rendering. Příklady operací, které přidávají mikroúkoly do fronty, jsou operace související s Promise jako jsou `Promise.resolve().then()` a `Promise.reject().catch()` nebo zpětná volání z MutationObserver API.

Mikroúkolová fronta má vyšší prioritu než makroúkolová. To znamená, že pokud jsou nějaké mikroúkoly k vykonání, JavaScript bude nejprve zpracovávat tyto mikroúkoly a až po jejich dokončení přejde k makroúkolům. Toto uspořádání zajišťuje, že mikroúkoly jsou vykonány co nejdříve a umožňuje JavaScriptu efektivněji řídit asynchronní operace a zajistit plynulost uživatelského rozhraní.

3.3 Metody pro práci s event loop

3.3.1 requestAnimationFrame

`requestAnimationFrame` je metoda v JavaScriptu, která požádá prohlížeč o naplánování aktualizace animace. Tato metoda zpravidla zajišťuje, že JavaScriptový kód pro animaci se spustí před dalším překreslením prohlížeče. Při použití `requestAnimationFrame` je zpětné volání zařazeno do fronty událostí v okamžiku, kdy prohlížeč plánuje další překreslení. Prohlížeč může optimalizovat animaci tak, že vykoná několik requestAnimationFrame callbacků ve stejném rámci, minimalizuje nebo eliminuje vizuální artefakty a může zastavit animaci, když je okno neaktivní.

requestAnimationFrame je speciální typ asynchronní operace, který je specificky navržen pro animace a hry. Není to přímo součástí makroúkolové nebo mikroúkolové fronty, ale je součástí vlastního renderovacího cyklu prohlížeče, který se obvykle děje 60 krát za sekundu pro hladké animace. Použití requestAnimationFrame je obecně preferováno pro animace a hry, oproti použití setTimeout nebo setInterval, protože je lépe synchronizováno s překreslovacím cyklem prohlížeče a může vést k hladší a efektivnější animaci.

```
1 function animate() {
2   // kód pro animaci zde
3   requestAnimationFrame(animate);
4 }
```

```
5 requestAnimationFrame(animate);
```

Zdrojový kód 4: Příklad requestAnimationFrame

V příkladu 3.3.1 je funkce animate, která obsahuje kód pro jeden snímek animace. Po výpočtu tohoto snímku se funkce requestAnimationFrame zavolá znovu, což vede k tomu, že se celý proces opakuje. Je důležité poznamenat, že requestAnimationFrame není podporována ve všech prohlížečích, takže je dobré používat polyfill pro zajištění kompatibility napříč prohlížeči. [15]

3.3.2 setTimeout

setTimeout je globální funkce v JavaScriptu, která umožňuje nastavit časový interval (v milisekundách), po kterém se má vykonat určitá funkce nebo kód. setTimeout je funkce vyššího řádu, což znamená, že přijímá jinou funkci jako argument. Tato funkce se pak spustí po uplynutí určitého časového intervalu, který je také poskytnut jako argument. [16]

```
1 function greet() {
2   console.log("Hello, World!");
3 }
4 setTimeout(greet, 2000);
```

Zdrojový kód 5: Příklad setTimeout

Nicméně, setTimeout nezastavuje nebo nepozastavuje vykonávání zbytku kódu. Místo toho se zaregistruje událost, která se má stát v budoucnosti, a pak se pokračuje v dalším vykonávání kódu. Funkce se tedy typicky zavolá později než po

Zrušení timeoutu setTimeout vrací „timeout ID“, které může být použito ke zrušení timeoutu pomocí funkce clearTimeout. V příkladu 3.3.2 se funkce greet nikdy nespustí, protože timeout byl zrušen pomocí clearTimeout předtím, než mohl uplynout časový interval.

```
1 const timeoutID = setTimeout(greet, 2000);
2 clearTimeout(timeoutID);
```

Zdrojový kód 6: Zrušení timeoutu

3.3.3 setInterval

setInterval je funkce v JavaScriptu, která se používá k opakovanému spouštění určité funkce v daném časovém intervalu. Tato funkce se pokračuje v provádění, dokud ji nezastavíte pomocí funkce clearInterval nebo dokud stránka nebo aplikace neběží. V příkladě 3.3.3 se "Hello, World!"vypíše do konzole každé dvě

sekundy (2000 milisekund). Podobně jako setTimeout, i setInterval vrací ID intervalu, které může být použito k zastavení intervalu pomocí funkce clearInterval. V příkladě 3.3.3 je interval zastaven pomocí clearInterval, takže funkce greet se po prvním spuštění nebude opakovat. [17]

```
setInterval(function () {
  console.log("Hello, World!");
}, 2000);
```

Zdrojový kód 7: Příklad setInterval

```
const intervalID = setInterval(greet, 2000);
clearInterval(intervalID);
```

Zdrojový kód 8: Zrušení intervalu

3.3.4 Promise

Promise je objekt v JavaScriptu, který je používán pro asynchronní výpočty. Promise je obvykle používán k ošetření asynchronních operací, jako jsou požadavky na server pomocí AJAX. Promise vrací hodnotu, kterou je možno použít, až bude asynchronní operace dokončena nebo selže. [9] Promise se vždy nachází v jednom ze tří stavů:

- Pending (čekání): Iniciální stav Promise, který není ani splněný, ani odmítnutý.,
- Fulfilled (splněný): Znamená, že operace byla úspěšně dokončena.,
- Rejected (odmítnutý): Znamená, že operace selhala.

3.4 Monkey patching

Monkey patching je programovací termín, který se vztahuje na dynamické (nebo běžící) změny tříd nebo modulů. To se děje po načtení kódu do paměti. Monkey patching je silný nástroj, který umožňuje upravit nebo rozšířit chování knihoven, modulů, tříd nebo metod za běhu, aniž by bylo nutné měnit samotný zdrojový kód. [18]

Formální popis Nechť M je modul nebo třída a f je funkce nebo metoda v M. Monkey patching znamená změnu f v M tak, že se nahradí novou funkcí nebo metodou f'. To je možné díky dynamické povaze některých programovacích jazyků, jako jsou Python, JavaScript nebo Ruby. Změna f na f' je prováděna za běhu programu a ovlivňuje všechny budoucí volání f v M.

Přestože Monkey patching může být užitečný nástroj pro rychlé opravy nebo testování, měl by být používán s opatrností, protože může vést k nečekaným

chováním a těžko odhalitelným chybám, zejména pokud se provádí na třídách nebo modulech, které nejsou určeny k úpravám.

Monkey patching v JavaScriptu možňuje rozšířit nebo měnit chování existujících objektů, včetně vestavěných nativních objektů, uživatelsky definovaných objektů nebo hostitelských objektů poskytnutých prostředím, jako je prohlížeč. Tento proces se provádí přiřazením nové funkce na existující vlastnost objektu, jako v kódu 3.4. V tomto příkladu jsme upravili metodu hello prototypu String. Původně tato metoda vracela řetězec "Hello, World!", ale po aplikaci monkey patching nyní vrací "Hello, Universe!".

```
1  // Původní funkce
2  String.prototype.hello = function () {
3    return "Hello, World!";
4  };
5  // Použití původní funkce
6  console.log("test".hello()); // vypíše "Hello, World!"
7  // Monkey patching
8  String.prototype.hello = function () {
9    return "Hello, Universe!";
10 };
11 // Použití upravené funkce
12 console.log("test".hello()); // nyní vypíše "Hello, Universe!"
```

Zdrojový kód 9: Příklad monkey patching v JavaScriptu

Je důležité si uvědomit, že i když je monkey patching silný nástroj, měl by být používán s opatrností. Mohl by způsobit nečekané chování nebo konflikty, pokud se stejná metoda nebo vlastnost objektu upravuje na více místech. To je obzvláště důležité v JavaScriptu, kde mohou být objekty a jejich prototypy sdíleny napříč různými částmi kódu nebo dokonce různými knihovnami.

3.5 Iframe

Iframe, neboli vložený rámec, je HTML dokument vložený do jiného dokumentu na webové stránce. Element iframe umožňuje webové stránce zobrazit obsah z jiného zdroje s vlastním posuvníkem, nezávisle na okolní stránce.

Obsah třetích stran Když je obsah vložený v iframe z domény odlišné od domény hlavní stránky, hovoří se o „iframe třetí strany“. Toto se často využívá pro zahrnutí obsahu, který je poskytován a hostován třetí stranou. Iframe třetí strany se široce používají pro vkládání reklam, videí (například vložení z YouTube), widgetů, obsahu sociálních médií (jako tlačítka Líbí se na Facebooku) nebo formulářů (například Google Formuláře) do webové stránky.

Prohlížeče, jako je Google Chrome, používají technologii zvanou „sandboxing“ k oddělení těchto Iframů a dalších procesů pro zvýšení bezpečnosti. Tento proces může zabránit škodlivým skriptům v iframe od přístupu k citlivým informacím na hlavní stránce. [19] [20] [21]

Iframe umožňuje integraci externího obsahu bez nutnosti hostování na hlavním webu. Také poskytuje určitou míru oddělení mezi hlavním webem a vloženým obsahem, což může být užitečné z hlediska bezpečnosti a správy obsahu.

Existují určité bezpečnostní a soukromí týkající se aspekty iframe třetích stran. Vzhledem k tomu, že obsah pochází z jiné domény, mohou vzniknout problémy jako cross-site scripting (XSS) nebo sledování uživatelů třetími stranami. Weboví vývojáři často musí zvážit tyto aspekty při integraci iframe třetích stran do svých stránek.

Z hlediska výkonu iframe třetí strany přináší určitou režii. Pokud je na stránce příliš mnoho iframů, může to zpomalit načítání stránky a zvýšit využití paměti.

Iframe z pohledu přehrávače Pro přehrávání obsahu je iframe ideálním prvkem, odděluje kontext od hlavní stránky. Přehrávač při inicializaci vytváří iframe element, do něhož poté vytvoří nahraný DOM. Tento iframe je izolovaný z hlediska stylů, ale sdílí s hlavním (rodičovským) oknem prostředky – paměť i proces. Do tohoto iframe elementu lze tedy přistupovat přímo, lze manipulovat s jeho elementy stejně, jako by to byly elementy hlavního okna.

3.6 Virtuální DOM

Virtuální DOM je vrstva mezi objektovým modelem klienta a kódem aplikace. Využívá se ve frameworcích jako React, Vue a ELM. Virtuální DOM je programový koncept, kde je zamýšlená reprezentace uživatelského rozhraní uložena v paměti a synchronizována s reálným objetkovým modelem pomocí knihovny, jako je například ReactDOM. [22]

Virtuální DOM funguje následujícím způsobem. Když dojde ke změně v uživatelském rozhraní, celý UI je znovu vykreslen v paměti pomocí virtuálního DOM. Poté se nový virtuální DOM porovná s původním snapshotem, který byl uložen před aktualizací. Tento proces se nazývá „diffing“. Jakmile knihovna zjistí, které objekty se změnily, aktualizuje reálný DOM pouze těmi objekty a částmi, které byly ve skutečnosti změněny. Tento proces se nazývá „reconciliation“.

Výhodou tohoto přístupu je, že manipulace s DOM jsou jedny z nejnáročnějších operací v prohlížeči, a omezit je může vést ke značnému zvýšení výkonu. To je obzvláště důležité v JavaScriptových aplikacích, kde je třeba často aktualizovat uživatelské rozhraní.

Zároveň je důležité poznamenat, že virtuální DOM může být méně efektivní pro některé úkoly a aplikace, které vyžadují intenzivní manipulaci s DOM. [23]

3.6.1 Virtuální DOM z pohledu DX

Virtuální DOM přináší výhody zejména pro vývojáře, ti se totiž (teoreticky) nemusejí starat o změny webu, naopak popisují stavy, do kterých se má web dostat a o zbytek se v ideálním případě postará framework. V praxi je však potřeba funkci používané knihovny dobře znát, alespoň základní znalost principů

pomáhá předcházet vytváření chyb a zároveň je lépe odhalovat. Virtuální objektový model přináší také zlepšení výkonu, které ale může se špatným přístupem naopak obrátit a výkon zhoršit.

Deklarativní syntaxe VDOM umožňuje vývojářům psát kód v deklarativní syntaxi, což znamená, že místo toho, aby se museli starat o to, jak a kdy aktualizovat DOM, jednoduše popíšou, jak by měl vypadat výsledný UI ve vztahu k danému stavu aplikace. Framework pak zařídí, aby se skutečný DOM shodoval s tímto popisem.

Výkonnost VDOM umožňuje efektivnější aktualizaci DOM, neboť místo přímé manipulace s reálným DOM, která může být výpočetně náročná, se změny nejprve provedou na virtuálním DOM. Poté se vypočte rozdíl (tzv. diffing) mezi aktuálním a novým VDOM a do skutečného DOM se provedou pouze minimálně potřebné změny (tzv. reconciliation). To může vést ke značnému zvýšení výkonnosti aplikace.

Univerzálnost Protože VDOM je jen JavaScriptový objekt, může být vykreslen nejen do webového prohlížeče, ale i do jiných prostředí, jako je mobilní aplikace (React Native), desktopové aplikace (Electron) nebo dokonce do formátu pro tisk. S VDOM lze pracovat i ve Web workers 3.2.1.2.

Testovatelnost VDOM může být snadno vytvořen a porovnán v rámci testů bez nutnosti manipulace s reálným DOM nebo dokonce bez nutnosti běžícího prohlížeče. To umožňuje psát rychlejší a spolehlivější jednotkové testy.

Přestože práce s VDOM přináší mnoho výhod, je také důležité si uvědomit, že některé operace mohou být v rámci VDOM náročnější, jako například přímá manipulace s DOM elementy nebo zacházení s focus managementem. [23]

3.6.1.1 Úskalí virtuálního DOM

I když virtuální DOM přináší mnoho výhod, existují i některé potenciální nevýhody nebo komplikace, které mohou vzniknout při jeho použití. VDOM může v určitých případech přinášet velkou režii a snižovat tak výkon aplikace. Kromě toho může být problém v kompatibilitě s existujícím kódem, nebo v přidané složitosti. [24]

Výkon Ačkoli VDOM obecně zvyšuje výkon tím, že minimalizuje manipulaci s reálným DOM, vytváření a porovnávání VDOM stromů stále vyžaduje určité výpočetní zdroje. V extrémních případech, kdy je třeba často a rychle aktualizovat velké množství prvků, může být výkon VDOM omezen.

Kompatibilita s existujícím kódem Pokud již existuje JavaScriptový kód, který přímo manipuluje s DOM, může být jeho integrace s VDOM náročná. VDOM totiž očekává, že veškeré změny v DOM proběhnou skrze něj, takže přímé změny v DOM mohou způsobit nesrovnalosti.

Složitost Práce s VDOM může přidat dodatečnou složitost do kódu. Je třeba se naučit, jak pracovat s konkrétním frameworkem, který VDOM používá, a porozumět, jak správně strukturovat a aktualizovat svůj VDOM strom. Vyžaduje to pochopení konkrétní knihovny, která se při implementaci používá.

Optimalizace Některé optimalizace, které by byly možné při přímé práci s DOM, mohou být obtížnější nebo nemožné při použití VDOM. Například, pokud je potřeba přesně ovládat, kdy a jak se vykonávají určité DOM operace pro dosažení nejvyšší možné rychlosti nebo efektivity, může být toto omezeno abstrakcí, kterou VDOM představuje.

Práce s určitými API Některé webové API, jako je například focus management nebo přímé měření a manipulace s rozměry elementů, mohou být obtížnější v kontextu VDOM, kde není přímý a okamžitý přístup k reálným DOM elementům.

3.6.1.2 Existující implementace

Virtuální DOM je populární nástroj, tudíž existuje mnoho teoreticky použitelných implementací. Každá abstrakce však přináší určité omezení, které je při reprodukování obsahu omezující. Většina knihoven je navržena pro vytváření aplikací, typicky nepokrývají kompletní API, které je potřeba reprodukovat. Dalším problémem je nutnost modifikovat práci s elementy, výchozí implementace přehrávače manipuluje s elementy přímo. Z těchto důvodů je možnost použít existující implementaci omezena.

RRWeb Implementace konkurenčního nástroje rrweb [25] ve svém přístupu používá zrcadlení API prohlížeče – `Document`, `Node`, `Element` a další rozhraní používaná v DOM API používá po svém jako `IRRDocument`, `IRRNode`, `IRRElement` a podobně. Tato rozhraní jsou duplikována z DOM API, jen jsou v nich nahrazeny navazující typy.

Tato rozhraní jsou implementována v třídách `BaseRR*`. V navazujícím kódu se nahrané elementy rekonstruují do této struktury a tato struktura se poté porovnává se skutečným objektovým modelem, na který se aplikují změny.

Tento přístup umožňuje pracovat s virtuálním objektovým modelem úplně stejně, jako se skutečným objektovým modelem. V požadovaný okamžik se provede porovnání a aplikují se nutné změny. Díky tomu, že Document obsahuje factory metodu createElement, stačí pro aktivaci této funkcionality nahradit instanci `Document` za vlastní (`RRDocument`).

```
1 const document = new RRDocument();
2 const html = document.createElement("html");
3 const head = document.createElement("head");
4 const body = document.createElement("body");
5 html.append(head, body);
6 document.append(html);
```

Zdrojový kód 10: RRWeb VDOM příklad

Tuto implementaci lze shrnout jako zajímavou a lze se jejím přístupem inspirovat. Použití této knihovny však z praktických důvodů není vhodné. Vzhledem k nevelkému rozsahu implementace si tuto funkcionalitu můžeme vytvořit sami a mít všechno plně pod kontrolou. RRWeb se odlišuje přístupem k elementům jako je například canvas, nebo iframe a to by přinášelo problémy.

3.7 Vývojářské nástroje

Vývojářské nástroje v prohlížečích jsou určeny pro testování, ladění a monitorování webových aplikací. Tyto nástroje jsou integrovány přímo do webových prohlížečů a poskytují řadu funkcí, které pomáhají vývojářům optimalizovat a vylepšovat své aplikace. Mezi tyto nástroje patří například inspektor DOM, konzole, debugger, monitor síťové komunikace nebo profilování výkonu. Profilování obsahuje funkcionality k vyhodnocování výkonu aplikace. [6]

3.7.1 Profilování výkonu

Profiling v prohlížeči je proces sběru a analýzy dat o chování webové stránky nebo aplikace za účelem optimalizace výkonu. To zahrnuje sledování a měření času načítání, spotřeby paměti, a dalších metrik. Profiling může identifikovat úzká hrdla a pomoci vývojářům vylepšit efektivitu kódu a zrychlit načítání stránek. Některé prohlížeče poskytují nástroje pro vizualizaci a analýzu profilovacích dat, což umožňuje detailní pohled na běh webových aplikací.

4 Analýza stávajícího stavu

Webový přehrávač ve výchozím stavu reprodukuje nahraný obsah, dělá to ale v podstatě naivně. Vznikají v něm tedy problémy popsané v této sekci. V dalších odstavcích jsou popsány problémy ovlivňující uživatelský zážitek – souvislý výpočet, sdílené výpočetní prostředky a nadbytečné renderování

Některé problémy jsou demonstrovány na přiloženém kódu, který je po spuštění dostupný na adrese `http://localhost:3000`.

4.1 Souvislý výpočet

Synchronní výpočet popsaný v 3.2.1 způsobuje, že při vykonávání kódu delším než je jeden snímek, může uživatel zpozorovat zpožděné reakce stránky. Délka snímku je dána obnovovací frekvencí monitoru. Tento problém je možno pozorovat na přiložené aplikaci na adrese `/rozdeleni`. Při experimentování s délkou rámce lze pozorovat zpožďování, či zamrzání animace. V přehrávači tento problém velmi ovlivňuje uživatelský zážitek, lze pozorovat zpožděné překreslování i zpomalené reakce na vstupy.

Souvislý výpočet v přehrávači vzniká při modifikaci struktury objektového modelu dokumentu. V závislosti na okolnostech mohou vznikat výpočty dlouhé několik vteřin. S takovým omezením není možné aplikaci možné aplikaci vydat do produkčního prostředí.

4.2 Sdílené výpočetní prostředky

Přehrávač sdílí výpočetní prostředky s aplikací, do které je integrován. To může prodlužovat některé snímky a zhoršovat uživatelský zážitek.

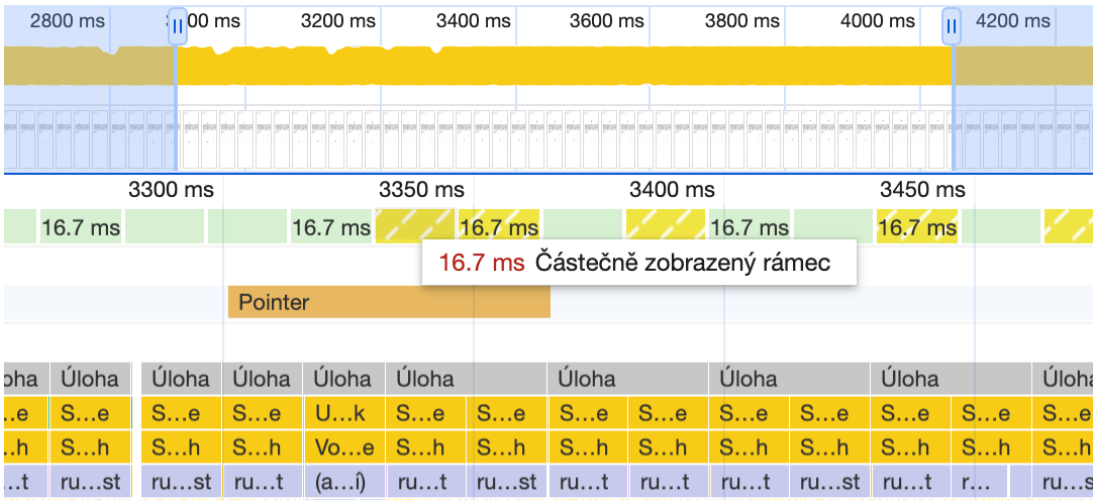
Tento problém si lze představit jako skládání více rámců dohromady. Přestože je výpočet omezen na krátký interval, bude-li podobný výpočet iniciován z více zdrojů (například z aplikace a přehrávače) současně, tedy v jednom rámci, nestihne prohlížeč v očekávaném čase překreslit.

Pro reprodukci tohoto problému slouží stránka `/iframe`. I přes malou délku rámce lze vícenásobným kliknutím na tlačítko animaci zpomalit. Tento problém lze pozorovat na 3. Zde se requestAnimationFrame volá v jednom procesu v každém snímku dvakrát po 16 ms a to blokuje hlavní vlákno vždy na 32 ms.

4.3 Nadbytečné renderování

Rozebíraný přehrávač provádí nahrané DOM mutace přímo na objektový model dokumentu. Tento přístup přináší problém při přepínání nahrávek a také při změně času v přehrávači. Problém je demonstrován na adrese `/vdom`, je třeba ponechat prázdné políčko VDOM. Lze experimentovat s počtem uzlů, pro malý počet uzlů je problém nevýrazný, ale s počtem uzlů se lineárně zvětšuje. Demonstrativní příklad funguje tak, že vytvoří zadaný počet uzlů při každém

Obrázek 3: Spojené výpočetní prostředky



stisku tlačítka. To simuluje přechod mezi nahrávkami, kde se v každém přechodu znovu-vytváří struktura stránky, která je většinou velmi podobná.

4.3.1 Přepínání nahrávek

Přepnutí nahrávky znamená často smazání a vytvoření velmi podobné struktury DOM. Znamená to nejen zdržení, ale také vizuální odpad – částečně vyrenderovanou stránku, která může mít v různých fázích rekonstrukce odlišnou strukturu. Tuto „stránku v rekonstrukci“ je třeba překrýt vizuálním prvkem, aby uživatele během používání nerušila.

4.3.2 Změna času

Při změně času se může přehrávat velký počet mutací v jednom okamžiku. To zahrnuje i mutace elementů, které se v tomto okamžiku vytvoří a záhy se zase smažou, protože ve výsledném čase již nejsou. Dobrým příkladem jsou aplikace se stránkováním, obsah stránkování je opakovaně nahrazován a v přehrávači se tak může zbytečně vykreslit velký počet elementů, jež jsou následně smazány.

4.4 Zdlouhavé vydávání nových verzí

Vydávání nových verzí softwaru je běžným a nezbytným procesem, který se pravidelně vyskytuje v rámci vývojových cyklů. Tento proces je zásadní pro zlepšení a udržování kvality softwaru, ať už se jedná o přidávání nových funkcionalit nebo opravy existujících chyb.

Přidávání nových funkcionalit je klíčovou součástí vývoje softwaru. Vývojáři neustále hledají způsoby, jak vylepšit své produkty a poskytnout uživatelům nové a inovativní funkce. Tyto aktualizace mohou zahrnovat nové nástroje, vylepšené

uživatelské rozhraní, integraci s jinými platformami nebo jakékoliv jiné funkce, které zlepšují celkovou funkčnost a uživatelský zážitek.

Opravy chyb jsou dalším zásadním prvkem v procesu vydávání nových verzí. Bez ohledu na to, jak pečlivě je software vyvinut, chyby jsou nevyhnutelné. Mohou se objevit v důsledku lidských chyb, nedostatků v původním designu nebo nečekaných interakcí mezi různými částmi softwaru. Pravidelné opravy těchto chyb nejenže zlepšují spolehlivost a stabilitu softwaru, ale také pomáhají zabezpečit, že uživatelé jsou chráněni před potenciálními bezpečnostními hrozbami.

Vydávání nových verzí také poskytuje vývojářům příležitost k získání zpětné vazby od uživatelů. Uživatelé mohou testovat nové funkce a poskytnout cennou zpětnou vazbu, která může vést k dalším vylepšením. Z tohoto důvodu je vydávání nových verzí nezbytným a neustálým procesem v rámci vývoje softwaru, jehož cílem je neustálé zlepšování a inovace.

Přehrávač je distribuován prostřednictvím soukroméno NPM repozitáře. Odtud je stahován do aplikace Smartlook. Tato aplikace je potom sestavena a nasazena do různých prostředí. Toto je výchozí řešení, které bylo přímočaré pro implementaci, má však nevýhody. Pro aktualizaci přehrávače je třeba sestavit a vydat celou aplikaci. Vyžaduje to synchronizaci mezi týmy a prodlužuje vydávání nových verzí.

K aktualizaci přehrávače jsou potřeba následující kroky.

- nahrání do npm repozitáře,
- aktualizace v aplikaci,
- sestavení aplikace,
- vydání aplikace

Z popsaných kroků je patrné, že aktualizace přehrávače vynucuje změny v aplikaci. Tento krok však není nezbytný a jeho vynechání by znamenalo výrazné zjednodušení a zrychlení celého procesu.

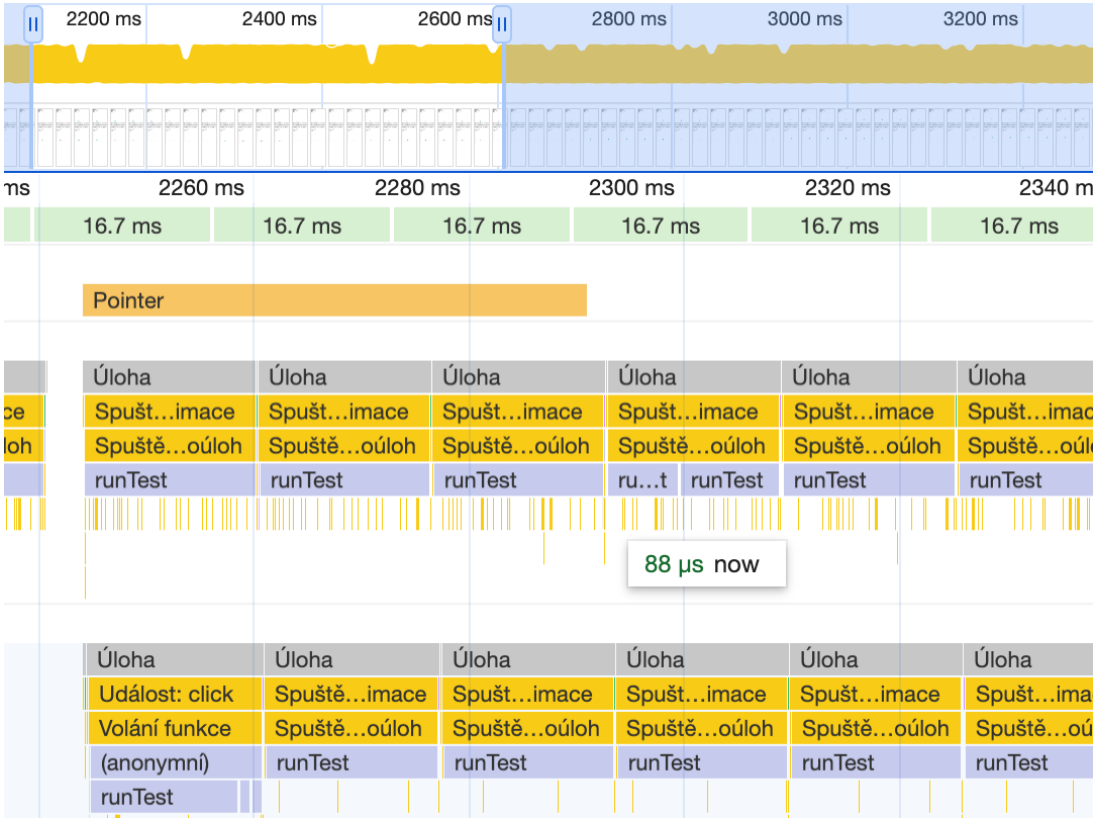
5 Popis vylepšení

5.1 Oddělení výpočetních prostředků

Výchozí implementace přehrávače sdílí výpočetní prostředky, tedy proces i paměť se stránkou, ve které je integrovaná. To znamená, že při vykonávání kódu přehrávače se vykonává ve stejném vlákně i kód stránky. Pokud přehrávač zamrzne, například z důvodu chyby v implementaci, zamrzne celá stránka.

Toto chování lze změnit použitím iframe třetí strany. Tento prvek přináší některé zajímavosti, zejména oddělené výpočetní prostředky. To znamená, že přehrávač může běžet v jiném procesu, než hlavní stránka a mít vlastní paměťový prostor. Pro realizaci tohoto vylepšení je třeba mít k dispozici další doménu, na kterou se přesune obsah iframe. Na obrázku 4 lze pozorovat vykreslování ve dvou procesech souběžně. Rámce nejsou řazeny za sebe jako na obrázku 3, ale každé pracuje ve svém sandboxu.

Obrázek 4: Oddělení výpočetních prostředků



5.1.1 Komunikace

Ke komunikaci s iframe třetí strany slouží `postMessage`, je tedy nutné přidat API, kde bude komunikace serializována a v iframe zase deserializována a pře-

dána přehrávači. Všechna data jsou ve formátu JSON a tak se při procesu serializace není třeba o nic starat. První část tohoto rozhraní znázorňuje kód 5.1.1. Na přiloženém kódu je vidět, jak přehrávač uvnitř iframe přijímá zprávy z rodičovského okna a na základě přijatého požadavku vykoná nějakou akci, nebo vrátí požadovanou informaci. Každý požadavek se potvrzuje zprávou poslanou zpět.

```
let player = null;

window.addEventListener("message", async (event) => {
  try {
    let returnValue;
    if (isCreateMessage(event.data)) {
      player = await initalizePlayer(event.data.config);
    } else {
      if (player === null) {
        throw new TypeError(
          "Player is not initialized. Call Player.create() first."
        );
      }

      if (isGetTimeMessage(event.data)) {
        returnValue = player.currentTime;
      } else if (isPauseMessage(event.data)) {
        player.pause();
      } else if (isPlayMessage(event.data)) {
        player.play();
      } else if (isSeekMessage(event.data)) {
        await player.seek(event.data.time);
      } else if (isSetEventsMessage(event.data)) {
        player.events = event.data.events;
      } else if (isSetSpeedMessage(event.data)) {
        player.speed = event.data.speed;
      }
    }

    window.parent.postMessage(
      createFinishedMessage(event.data, returnValue),
      "*"
    );
  } catch (error) {
    window.parent.postMessage(
      createErrorMessage(receivedEventData, String(error)),
      "*"
    );
  }
});
```

Zdrojový kód 11: Komunikace s iframe třetí strany ze strany přehrávače

Komunikace z opačné strany, tedy z aplikace do iframe, prezentuje kód 5.1.1. Je na něm demonstrováno aplikační rozhraní pro ovládání přehrávače, v metodě

`sendToIframe` je ukázáno čekání na návratovou zprávu z iframe.

```
export class OuterPlayer {
  static create = async (config) => {
    const player = new OuterPlayer();
    player.iframe = document.createElement("iframe");
    player.iframe.setAttribute("src", config.url);
    await new Promise((resolve) => {
      player.iframe.addEventListener("load", resolve);
    });
    await player.sendToIframe(createCreateMessage(config));
    return player;
  };

  getTime = () => this.sendToIframe(createGetTimeMessage());

  pause = () => this.sendToIframe(createPauseMessage());

  play = () => this.sendToIframe(createPlayMessage());

  sendToIframe = async (message) => {
    this.iframe.postMessage(message, "*");
    await new Promise((resolve) => {
      const callback = (event) => {
        if (isReturningMessage(message, event.data)) {
          resolve(event.data.returnValue);
          this.iframe.removeEventListener("message", callback);
        }
      };
      this.iframe.addEventListener("message", callback);
    });
  };

  setSpeed = (speed) => this.sendToIframe(createSetSpeedMessage(
      speed));

  seek = (time) => this.sendToIframe(createSeekMessage(time));
}
```

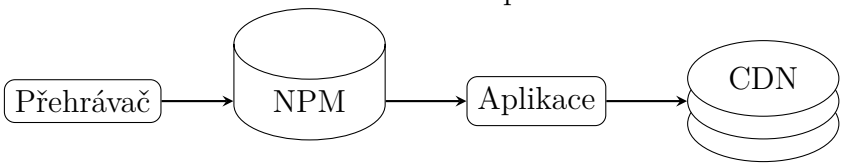
Zdrojový kód 12: Komunikace s iframe třetí strany ze strany aplikace

5.2 Nezávislé vydávání nových verzí

5.2.1 Distribuce přehrávače

Návrh vylepšení spočívá v rozdělení přehrávače na jednoduché aplikační rozhraní a přehrávač samotný. Aplikační rozhraní bude stejně jako doposud importováno do aplikace a sestavováno spolu s ní. Modul přehrávače se bude načítat z CDN až při přehrávání u klienta.

Obrázek 5: Distribuce přehrávače



5.2.2 Realizace

Pro realizaci tohoto vylepšení využijeme API popsané v kapitole 5.1. Ačkoliv by verze mohly být vydávány nezávisle i bez použití iframe třetí strany, obě vylepšení se vhodně doplňují. Obě implementace totiž vyžadují, aby byl přehrávač někde veřejně publikován a je proto výhodné jejich spojení. Musíme zvolit doménu tak, aby prohlížeč používal sandboxing. Aplikace běží na doméně app.smartlook.com, přehrávač tedy musíme umístit na jinou doménu druhého, nebo prvního řádu. Máme k dispozici doménu smartlook.cz, která těmto požadavkům skvěle vyhovuje.

Přehrávač je vydáván nezávisle na aplikaci, což zkracuje vydávací smyčku. Kratší vydávací smyčka zrychluje zejména opravu chyb a možnosti debugování. Do aplikace, ve které je přehrávač používán, se pomocí NPM repozitáře dostává jednoduché aplikační rozhraní, které dokáže načíst přehrávač ze vzdáleného zdroje a ten tak nemusí být součástí aplikace.

5.3 Rozdělení výpočtu

5.3.1 Dlouhý výpočet v přehrávači

Pro správu objektového modelu je nutné používat hlavní vlákno. V tomto vlákně prohlížeč také potřebuje renderovat a obsluhovat vstupy. Pro hladké fungování aplikace je tedy nutné si dát pozor na delší výpočty, které by mohly zabrat více než několik milisekund. Takovým výpočtem je ve zkoumaném přehrávači manipulace s DOM elementy. Pro komplexní weby o tisících elementech to může dělat i několik sekund v závislosti na použitém harware. V takových případech je nutné výpočet rozdělit. Jako kritérium k rozdělení se nabízí čas. Performance API poskytuje potřebnou funkcionalitu.

Výpočet lze rozdělovat mechanismy popsanými v sekci 3.2.2. Pro naše použití je ideální funkce `requestAnimationFrame`. Tu lze dobře kombinovat s Promise (3.3.4), generátory a iterátory (3.1.2). Základní provedení je v kódu 5.3.1.

```
1 const FRAME_LENGTH = 1000 / 60;
2 const NODES = [
3   // pole serializovaných uzlů
4 ];
5
6 const run = async () => {
7   let startTime = performance.now();
```

```
8   for (const node of render(NODES)) {
9     if (performance.now() - startTime > FRAME_LENGTH) {
10      await new Promise(requestAnimationFrame);
11      startTime = performance.now();
12    }
13    console.log("node zpracován", node);
14  }
15 };
16
17 function* render(nodes) {
18  for (const node of nodes) {
19    // náročná operace
20    yield node;
21  }
22 }
```

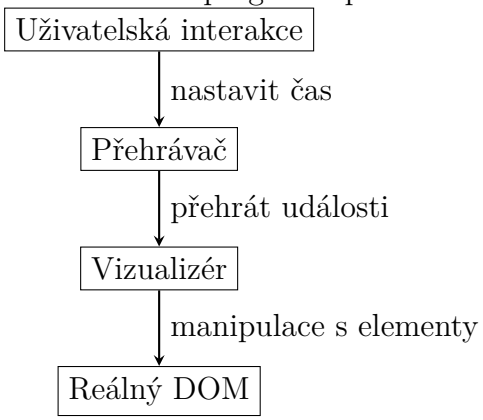
Zdrojový kód 13: Rozdělení výpočtu

5.4 Optimalizace manipulace s DOM

Optimalizace spočívá ve vynechání nadbytečných operací, konkrétně renderování struktury nahrávky, popsaných v kapitole 4.3. Tímto lze ušetřit cenné prostředky, které zpomalují interakci s přehrávačem. K této optimalizaci se nabízí použití virtuálního objektového modelu dokumentu popsaného v 3.6. Požadovaná struktura stránky se může v paměti porovnat s aktuálně vykreslenou strukturou a namísto celkové rekonstrukce se na reálný DOM aplikují pouze nutné změny.

5.4.1 Návrh

Obrázek 6: Tok programu při interakci

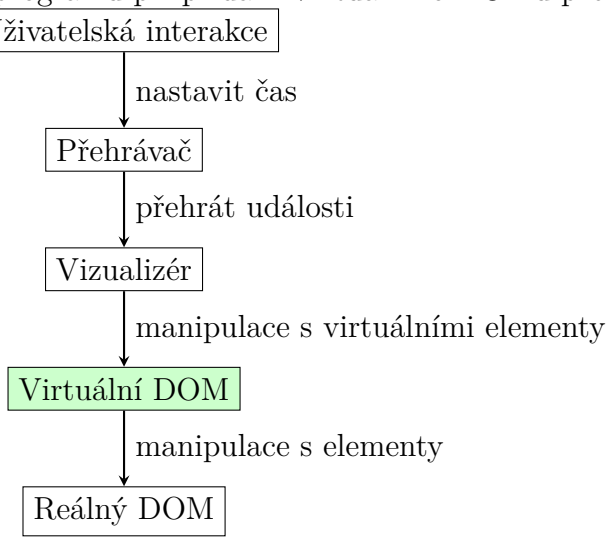


5.4.1.1 Nahrazení elementů za vlastní implementaci

Tato varianta má za cíl vytvořit abstrakci nad reálným DOM. K tomu je potřeba prakticky duplikovat rozhraní pro manipulaci s objektovým modelem podobně,

jako to dělá RRWeb 3.6.1.2.

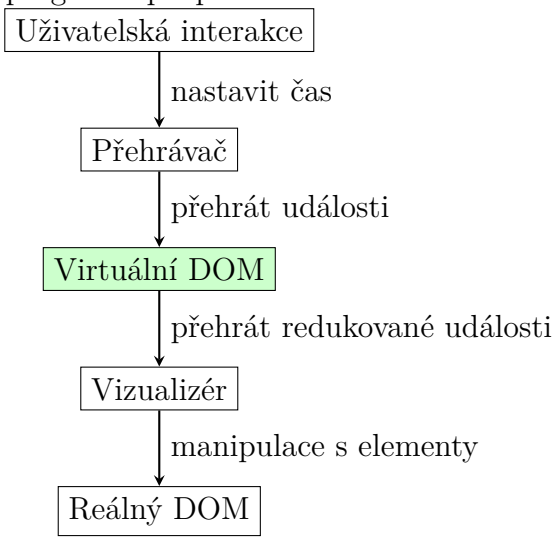
Obrázek 7: Tok programu při přidání Virtuálního DOMu před Reálný DOM



5.4.1.2 Redukce událostí

Druhou variantou je redukce událostí před odesláním do Vizualizéru. To znamená, že by vůbec nebylo potřeba upravovat manipulaci s DOM, k tomu se může používat stále stejný, dlouhodobě produkčně otestovaný kód. Benefitem tohoto řešení je odstínění od reálného DOM. To přináší teoretickou možnost kompletně rozdělit výpočet nad touto strukturou do vláken pomocí Web workers, popsaných v kapitole 3.2.1.2.

Obrázek 8: Tok programu při přidání Virtuálního DOMu před Vizualizér



5.4.2 Implementace

Implementace virtuálního DOM je součástí přílohy této práce. Jeho použití ve speciálním případě je demonstrováno na adrese `/vdom`. V příkladu je určitý počet uzlů, který se při použití virtuálního objektového modelu pouze porovná v paměti a ve výsledku jsou aplikovány pouze změny. Toto řešení se vyrovnává s použítím vnořených iframů

6 Použitá vylepšení

Popsaná vylepšení byla implementována v přehrávači Smartlook postupně a většina z nich je používána v produkčním prostředí. V této části jsou popsány dopady použitých vylepšení.

6.1 Rozdělení výpočtu do částí

Tímto vylepšením nastavujeme prohlížeči, jakou dobu může používat pro běh našeho kódu a dáváme mu „přestávky“ pro obsluhu uživatelských interakcí, které musí realizovat ve stejném vlákně. V tomto případě je používán ‘requestAnimationFrame‘, ale dle typu výpočtu lze k podobnému rozdělení použít i

Fundamentální věc, bez které by přehrávač nebylo možno vůbec seriozně používat. Na dělení výpočtu je třeba myslet při jakékoliv náročnější iteraci. Velkým pomocníkem jsou iterátory a generátory 3.1.2, které dovolují elegantně výpočet rozdělovat a doručovat mezivýpočty.

Rozdělení výpočtu přináší řadu výhod, zejména pro prohlížeč. Jednou z hlavních výhod je, že prohlížeč může „současně“ renderovat stránky a obsluhovat interakce uživatele. To znamená, že když uživatel prochází stránkou nebo s ní interaguje, prohlížeč může zároveň vykonávat výpočetní úkoly bez toho aby uživatele jakkoliv ovlivnil.

6.2 Iframe třetí strany

Toto vylepšení přináší rozdělení výpočetních prostředků do dvou procesů, primární proces obsluhuje hlavní stránku a sekundární se stará o akce v přehrávači. Oba procesy spolu komunikují prostřednictvím serializovaných zpráv. Ačkoliv je tedy JavaScript jednovláknový jazyk, tato funkcionalita přináší skutečnou paralelizaci.

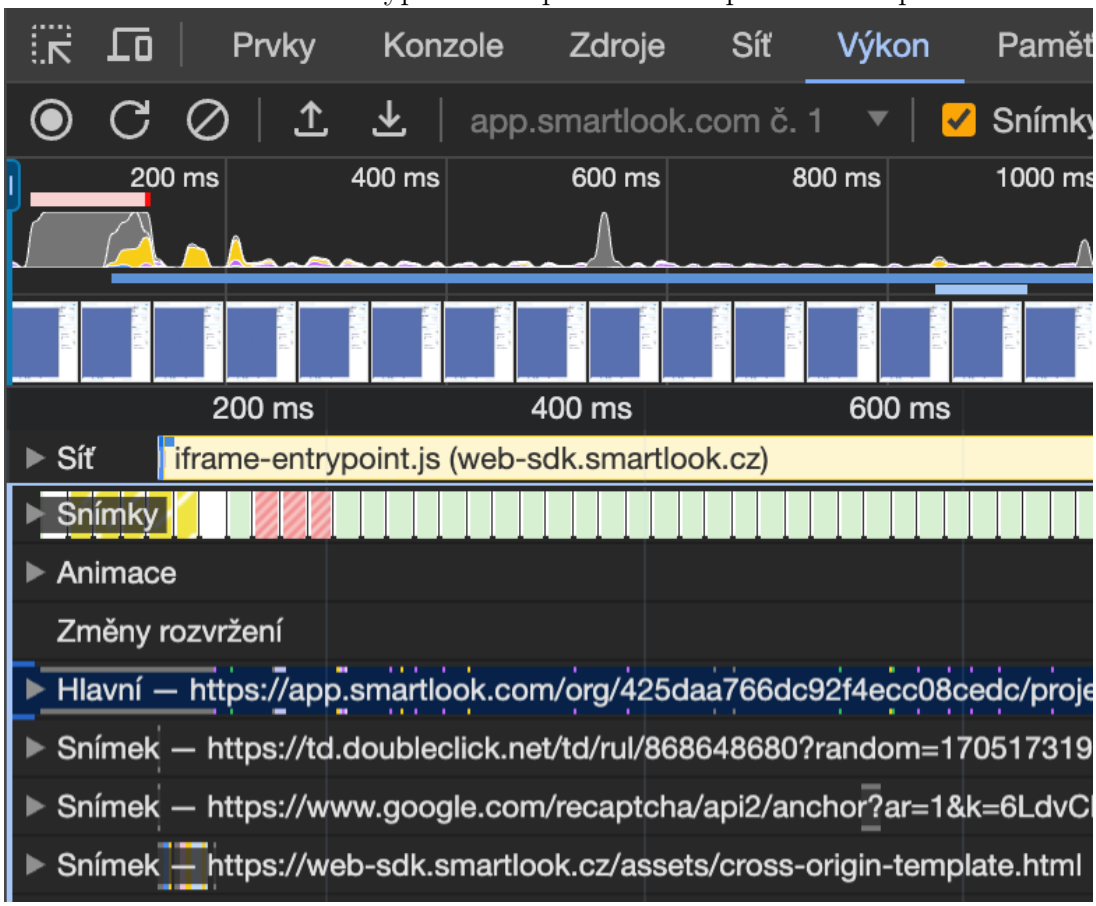
Iframe třetí strany byl nasazen na doméně smartlook.cz, přičemž aplikace běží na smartlook.com. Tuto skutečnost lze vyčíst i z obrázku 9, na kterém lze vidět oddělení procesů. Během vydání do produkčního prostředí se vyskytnul jeden problém popsaný dále.

Při vydání iframe třetí strany se vyskytnul problém, kdy některým klientům prvky zabezpečení blokovaly doménu smartlook.cz a přehrávač se kvůli tomu vůbec nenačetl. Tento případ je vyřešen kontrolou, zda se přehrávač podaří načíst do určitého času (několika vteřin). Pokud se to nepodaří, spustí se záložní řešení s iframe stejného původu a zároveň se tato informace uloží do lokálního úložiště, aby se při dalším načtení spustilo rovnou záložní řešení a nebylo třeba čekat.

6.3 Virtuální DOM

Tato funkcionalita přináší zajímavé výsledky, jak je patrné z obrázků níže. V rámci testování této funkcionality byla pořízena jednoduchá nahrávka jednostránkové aplikace na adrese github.com. Tato nahrávka byla poté otevřena v přehrávači

Obrázek 9: Oddělení výpočetních prostředků v produkčním prostředí

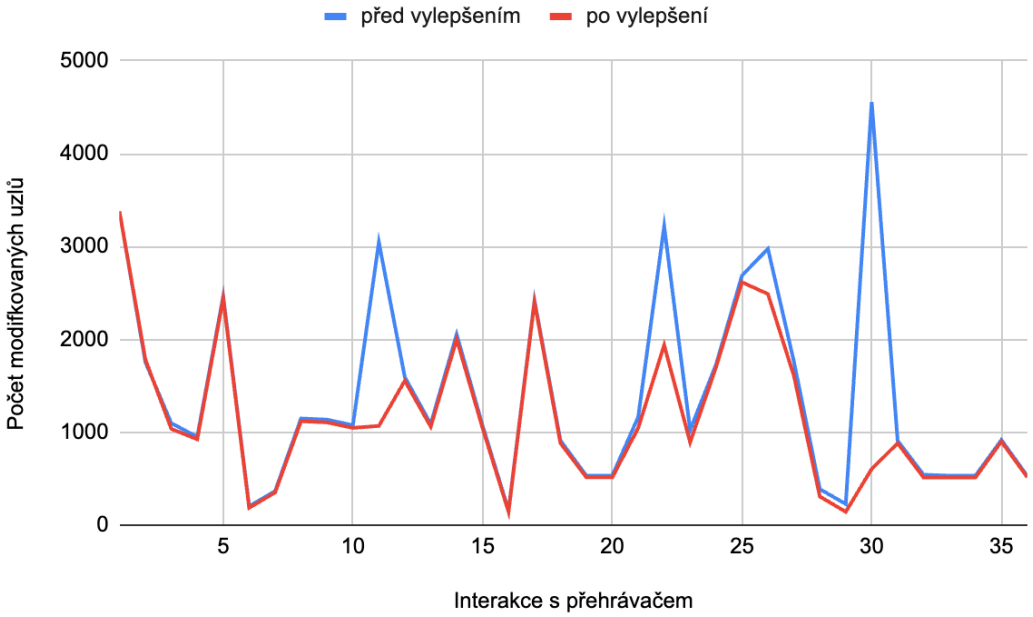


a byly na ni aplikovány 3 scénáře posunu času v nahrávce. Každý scénář byl spuštěn prostřednictvím konzole v přehrávači, jednou s vylepšením a podruhé bez něj. Prvním scénářem je lineární postup nahrávkou směrem vpřed, dalším scénářem je lineární postup směrem vzad a posledním scénářem je přeskakování ze začátku na konec nahrávky. Naměřená data ze scénářů jsou v Příloze A.

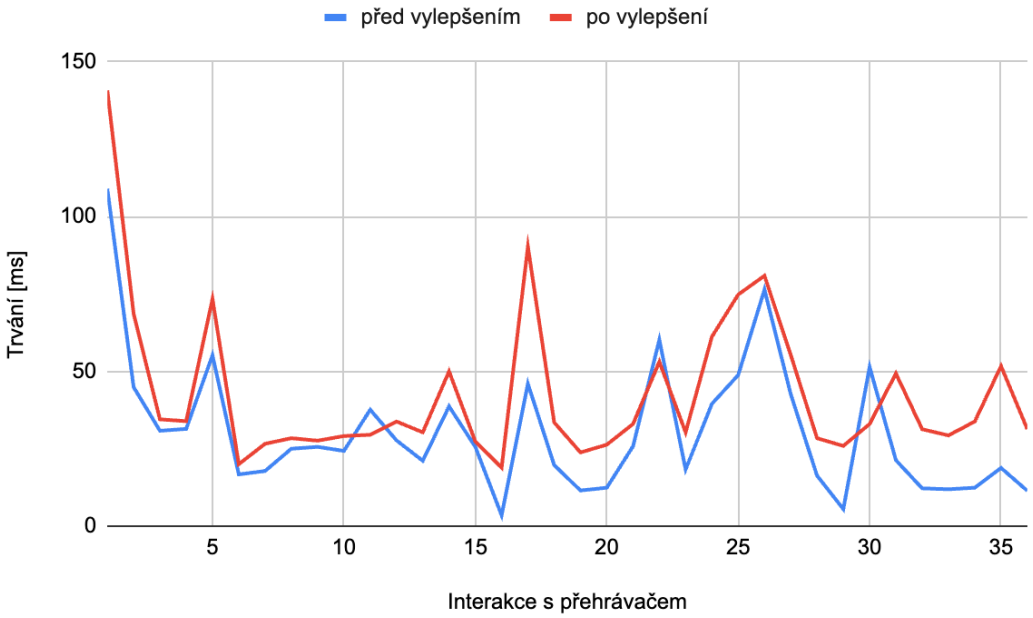
Z každého scénáře jsou tedy vytvořeny dva grafy prezentující chování přehrávače při interakci dané scénářem. První prezentuje počet mutací aplikovaných v přehrávači při změně času v nahrávce a druhý použitý čas v milisekundách. Na ose X je vždy zobrazen počet interakcí a na ose Y buď počet mutací, nebo spotřebovaný čas. Na obrázcích si lze všimnout, že křivka počtu mutací je podobná křivce trvání. Počet mutací v grafech znamená všechny sečtené mutace (přidávání, upravování a mazání) dohromady.

První scénář začíná na začátku nahrávky a posunuje se v čase směrem ke konci. To způsobuje postupnou aplikaci mutací, nevytvářejí se nadbytečné elementy. Záznam scénáře je vyobrazen na obrázcích 10 a 11. Na obou obrázcích je patrné, že křivky před i po vylepšení jsou velmi podobné, avšak z grafu trvání je vídět, že virtuální DOM přidává režii a v tomto konkrétním případě má mírně negativní dopady.

Obrázek 10: Počet mutací při navigaci v nahrávce směrem vpřed.



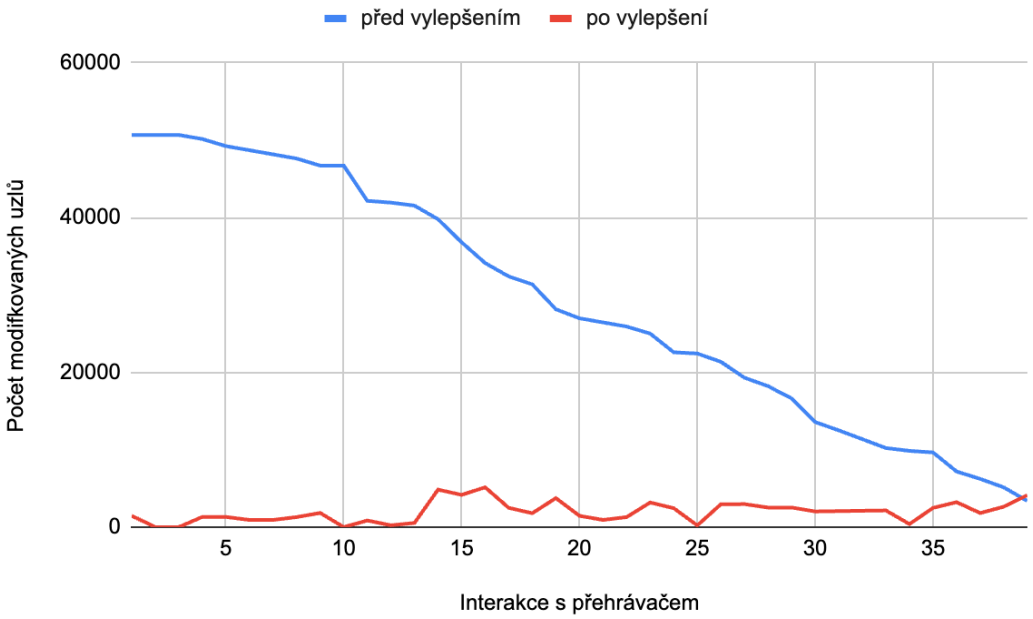
Obrázek 11: Délka trvání při navigaci v nahrávce směrem vpřed.



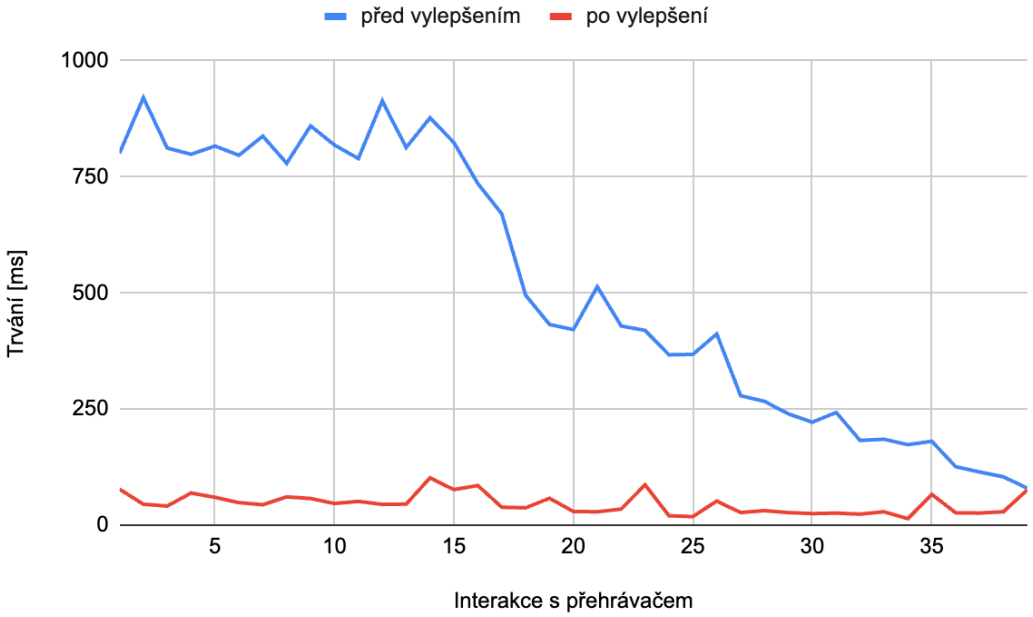
Ve druhém scénáři se přehrávač nejprve přesune na konec nahrávky a poté se v každém kroku posunuje směrem k začátku. To způsobuje v každém kroku přehrávání mutací od začátku nahrávky, proto lze na grafech vyobrazených na obrázcích 12 a 13 pozorovat na startu scénáře velký rozdíl, který se postupně

snižuje.

Obrázek 12: Počet mutací při navigaci v nahrávce směrem vzad.



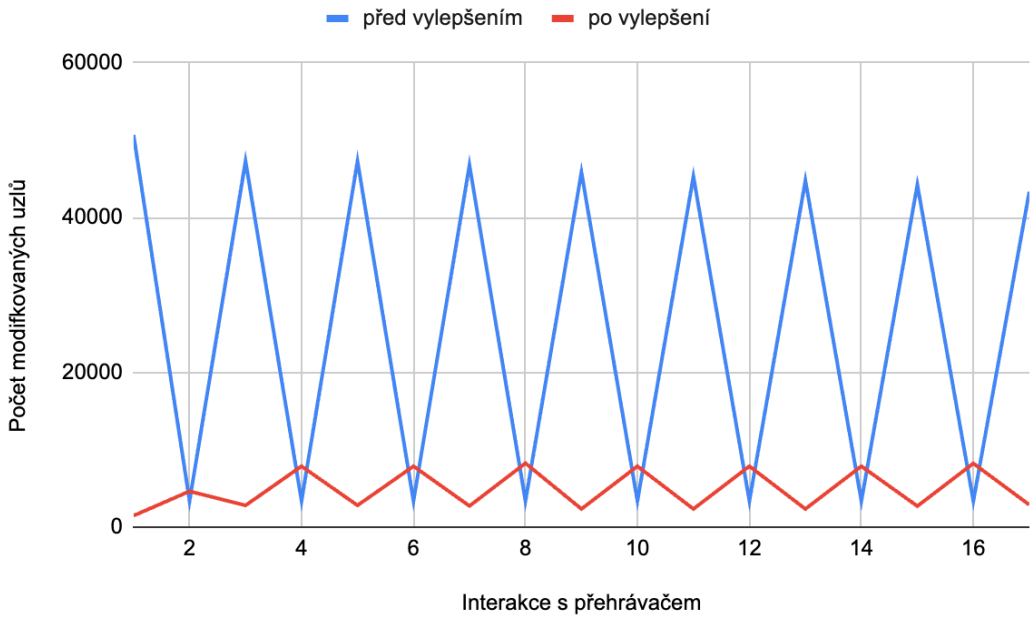
Obrázek 13: Délka trvání při navigaci v nahrávce směrem vzad.



Grafy posledního scénáře na obrázcích 14 a 15 ukazují rozdíly mezi přesunem na začátek a konec nahrávky. Je na nich patrné, jak nekonzistentně se přehrávač

před vylepšením chová. Změna času může trvat jednotku času, nebo také její desetinásobek, to je při používání aplikace matoucí.

Obrázek 14: Počet mutací při navigaci v nahrávce z jednoho konce na druhý, ze začátku na konec a zpět opakovaně.



Výsledky měření ukazují, že většinou virtuální DOM zlepšuje výkonnost aplikace. To znamená, že aplikace s virtuálním DOM obecně rychleji reagují na uživatelské interakce a efektivněji zpracovávají změny na stránce. Avšak výsledky také ukazují, že existují případy, kdy virtuální DOM nepřináší žádné výhody, a dokonce může výkon aplikace ovlivnit negativně. Proto je důležité při rozhodování o použití virtuálního DOM zvážit všechny klady a zápory a nespoléhat se na to, že virtuální DOM vyřeší všechny problémy s renderováním na webu. Jinými slovy, i když je virtuální DOM mocným nástrojem, není to všelék a mělo by být použito s opatrností a s přihlédnutím k konkrétnímu kontextu a potřebám aplikace.

Obrázek 15: Délka trvání při navigaci v nahrávce z jednoho konce na druhý, ze začátku na konec a zpět opakovaně.

Závěr

Tato práce se zabývá analýzou a vylepšením webové aplikace. V rámci tohoto tématu bylo identifikováno několik oblastí, které mohou být vylepšeny. Byla implementována vylepšení pro zachování responzivního chování aplikace i při zatížení procesoru. Prvním takovým vylepšením je rozdělení výpočtu na menší části. To dává prohlížeči možnost reagovat na uživatelské akce i během výpočtu. Druhým z těchto vylepšení je skutečná paralelizace výpočtu a vykreslování pomocí iframe třetí strany. Toto vylepšení využívá sandboxingu prohlížeče, který umožňuje spouštět kód v izolovaném prostředí.

Bylo představeno jedno vylepšení pro zrychlení vydávání nových verzí aplikace. Tímto jsme dosáhli zrychlení publikace nových verzí, což nyní trvá jen několik minut. Zrychlení dostupnosti nových verzí je důležité i pro opravy a hledání chyb. Dále byla implementována optimalizace pro minimalizaci spotřeby zdrojů. Spotřeba zdrojů je snížena pomocí optimalizace vykreslování. Implementovaný virtuální DOM způsobuje, že se překreslují pouze změněné části stránky. Na grafech je prezentováno, jak virtuální DOM snižuje množství manipulací s reálným DOM, ale také to, že v některých případech může být jeho použití nevýhodné. Celkově je však jeho použitím dosaženo snížení spotřeby zdrojů a snížení délky vykreslování. Implementovaný virtuální DOM je do budoucna možné zkusit využít i pro zrychlení výpočtu pomocí Web Workers.

V rámci práce byly představeny strategie pro řešení synchronizace výpočtu s vykreslováním prohlížeče. Tyto strategie mohou být použity jako inspirace pro řešení podobných problémů v libovolné jiné webové aplikaci. Zároveň byly tyto principy otestovány i v reálném, produkčním prostředí, kde obstály bez větších obtíží. Součástí práce jsou demonstrativní ukázky řešených problémů.

Conclusions

This thesis deals with the analysis and improvement of a web application. Several areas for improvement have been identified within this theme. Improvements have been implemented to maintain the responsive behavior of the application even when the processor is loaded. The first such improvement is to divide the calculation into smaller parts. This gives the browser the ability to respond to user actions even during the calculation. The second of these improvements is true parallelization of computation and rendering using third-party iframes. This enhancement uses browser sandboxing, which allows code to run in an isolated environment.

One improvement was introduced to speed up the release of new versions of the application. This has made it possible to speed up the publication of new versions, which now only takes a few minutes. Accelerating the availability of new versions is also important for fixes and bug-finding. Furthermore, optimization was implemented to minimize resource consumption. Resource consumption is reduced using rendering optimization. The implemented virtual DOM causes only the changed parts of the page to be redrawn. The graphs show how the virtual DOM reduces the amount of manipulations with the real DOM, but also that in some cases its use can be disadvantageous. Overall, however, its use results in reduced resource consumption and reduced rendering time. In the future, it is possible to try to use the implemented virtual DOM to speed up calculations using Web Workers.

As part of the thesis, strategies were presented for solving the synchronization of the calculation with the rendering of the browser. These strategies can be used as inspiration for solving similar problems in any other web application. At the same time, these principles were also tested in a real, production environment, where they held up without major difficulties. Demonstrative examples of solved problems are part of the thesis.

A Příloha

Příloha obsahuje tabulky s výsledky měření. V těchto tabulkách jsou obsažena data ze scénářů použitých pro metriky vylepšení pomocí virtuálního DOM. Tabulky obsahují následující sloupce:

- trvání [ms] - doba renderování v milisekundách
- událostí - počet událostí, které byly zpracovány
- přidáno - počet prvků, které byly přidány do DOM
- smazáno - počet prvků, které byly smazány z DOM
- upraveno - počet prvků, které byly upraveny v DOM

Tabulka 1: Scénář číslo 1 s vylepšením

trvání [ms]	událostí	přidáno	smazáno	upraveno
140.799999997019	2	3380	0	0
68.6000000014901	15	1666	69	48
34.5	18	961	70	5
33.8999999985098	20	849	70	5
73.3999999985098	19	2368	70	5
20	16	110	71	5
26.5999999940395	15	275	71	5
28.3999999985098	18	1050	65	4
27.6000000014901	17	1060	42	4
29.1000000014901	17	1002	43	2
29.5	19	1019	47	2
33.7999999970197	17	1509	44	2
30.3000000044703	16	1014	44	2
50	17	1955	44	2
27.3000000044703	16	998	44	2
18.8999999985098	16	89	48	6
90.2999999970197	19	2346	50	6
33.5	21	827	50	6
23.7999999970197	18	461	49	4
26.2999999970197	19	461	49	4
33	22	990	49	8
53.1000000014901	49	1882	51	6
30.3000000044703	23	837	50	5
61.2000000029802	27	1646	49	6
74.8000000044703	25	2568	46	2
80.8999999985098	26	2434	49	6
55.2999999970197	28	1563	44	2
28.3999999985098	17	274	25	9
25.8999999985098	17	107	17	19
33	20	557	34	14
49.2999999970197	19	827	49	6
31.2999999970197	20	461	49	3
29.3000000044703	18	461	49	3
33.8000000044703	18	461	49	3
51.7000000029802	18	847	49	4
31.2999999970197	15	461	50	4
30.3999999985098	1	0	0	1

Tabulka 2: Scénář číslo 1 bez vylepšení

trvání [ms]	událostí	přidáno	smazáno	upraveno
109	2	3380	0	1
44.8999999985098	15	1667	70	18
30.7999999970197	18	997	78	21
31.4000000059604	20	858	72	22
55.1999999955296	19	2369	71	25
16.7000000029802	16	111	72	20
17.8000000044703	15	276	72	18
25	18	1054	68	26
25.6000000014901	17	1066	45	25
24.2999999970197	17	1004	45	27
37.6000000014901	19	1997	1023	27
27.7000000029802	17	1517	52	25
21.0999999940395	16	1019	46	24
38.6999999955296	17	1956	45	53
25.6000000014901	16	999	45	27
3.5	16	90	49	19
46	19	2347	51	22
19.6999999955296	21	836	52	23
11.5	18	462	50	20
12.3999999985098	19	462	50	21
25.7999999970197	22	1078	60	27
60.1999999955296	49	3037	108	75
18.2999999970197	23	941	58	25
39.5	27	1647	50	33
48.7999999970197	25	2576	54	58
76.4000000059604	26	2849	70	57
42.6000000014901	28	1664	52	46
16.2999999970197	17	326	31	31
5.5	17	162	25	40
51.3999999985098	20	2637	1876	44
21.2999999970197	19	836	51	22
12.2000000029802	20	470	51	21
11.8999999985098	18	462	50	20
12.3999999985098	18	462	50	20
18.7999999970197	18	847	49	21
11.3999999985098	15	461	50	17
0.600000001490116	1	0	0	1

Tabulka 3: Scénář číslo 2 s vylepšením

trvání [ms]	událostí	přidáno	smazáno	upraveno
77.3999999985098	698	1477	0	0
45	697	0	0	1
41	697	0	0	0
69.1000000014901	682	847	461	4
59.7999999970197	664	461	847	4
48.3999999985098	646	461	461	3
43.8999999985098	628	461	461	3
60.7999999970197	608	827	461	3
57.2999999970197	589	995	827	6
46.5	589	0	0	0
51.1000000014901	569	278	557	16
44.6000000014901	552	107	107	18
45.2000000029802	535	250	274	8
101.80000000447	507	3277	1563	1
76.5	481	1725	2434	5
85.2000000029802	456	2572	2568	2
38.5	429	837	1646	5
37.1999999955296	406	955	837	4
57.8000000044703	357	1863	1882	2
29.2999999970197	335	461	990	7
28.6999999955296	316	461	461	3
34.5	298	827	461	3
86.7999999970197	277	2346	827	5
20.1000000014901	258	89	2346	5
18.3000000044703	242	125	89	5
51.7999999970197	226	1955	998	1
27	209	1014	1955	1
31.1000000014901	193	1509	1014	2
26.8999999985098	176	1014	1509	2
24.8000000044703	157	1002	1015	6
25.8000000044703	140	1060	1002	1
23.6000000014901	123	1049	1060	3
28.7000000029802	105	1093	1054	3
13.8000000044703	90	110	275	5
66.2999999970197	74	2367	110	5
26.5	55	848	2368	5
25.8999999985098	35	961	849	5
28.8000000044703	17	1666	961	5
75.8999999985098	2	2426	1666	48

Tabulka 4: Scénář číslo 2 bez vylepšení

trvání [ms]	událostí	přidáno	smazáno	upraveno
800.799999997019	698	44922	4741	1022
920.60000000149	697	44922	4741	1021
812.20000000298	697	44922	4741	1021
798.699999995529	682	44461	4691	1004
816.60000000149	664	43614	4642	983
796.60000000149	646	43152	4592	963
837.899999998509	628	42690	4542	943
779.399999998509	608	42220	4491	922
860	589	41384	4440	900
818.699999995529	589	41384	4440	900
789.60000000149	569	38747	2564	856
913.799999997019	552	38585	2539	816
813.20000000298	535	38259	2508	785
877.399999998509	507	36595	2456	739
823.899999998509	481	33746	2386	682
735.70000000298	456	31170	2332	624
670.70000000298	429	29523	2282	591
495.20000000298	406	28582	2224	566
432.199999995529	357	25545	2116	491
421.099999994039	335	24467	2056	464
513.60000000149	316	24005	2006	443
428.899999998509	298	23543	1956	423
419.40000000596	277	22707	1904	400
366.799999997019	258	20360	1853	378
367.699999995529	242	20270	1804	359
411.699999995529	226	19271	1759	332
278.70000000298	209	17315	1714	279
266.5	193	16296	1668	255
239.5	176	14779	1616	230
221.70000000298	157	12782	593	203
242.60000000149	140	11778	548	176
182.40000000596	123	10712	503	151
184.899999998509	105	9658	435	125
173.30000000447	90	9382	363	107
180.60000000149	74	9271	291	87
125.799999997019	55	6902	220	62
114.399999998509	35	6044	148	40
104	17	5047	70	19
79.5	2	3380	0	1

Tabulka 5: Scénář číslo 3 s vylepšením

trvání [ms]	událostí	přidáno	smazáno	upraveno
78.2999999970197	698	1477	0	0
103.799999997019	2	3241	1338	49
78.4000000059604	695	1338	1408	59
102.20000000298	2	3241	4579	49
64.8999999985098	695	1338	1408	59
112.299999997019	2	3241	4579	49
72.1000000014901	680	1724	947	59
94.7000000029802	2	3241	4965	49
64.2000000029802	662	1338	947	59
113.30000000447	2	3241	4579	49
63.3999999985098	644	1338	947	59
111.399999998509	2	3241	4579	49
66.1000000014901	626	1338	947	59
99.9000000059604	2	3241	4579	49
73.7000000029802	606	1704	947	59
110.5	2	3241	4945	49
74.3999999985098	587	1872	947	59
101.80000000447	2	3241	5113	49

Tabulka 6: Scénář číslo 3 bez vylepšení

trvání [ms]	událostí	přidáno	smazáno	upraveno
844.5	698	44922	4741	1022
74.8999999985098	2	3380	0	1
737.80000000447	695	41542	4741	1020
83.3999999985098	2	3380	0	1
754.299999997019	695	41542	4741	1020
87	2	3380	0	1
733.60000000149	680	41081	4691	1003
65.4000000059604	2	3380	0	1
674.80000000447	662	40234	4642	982
75.6000000014901	2	3380	0	1
663.5	644	39772	4592	962
65.2999999970197	2	3380	0	1
649.899999998509	626	39310	4542	942
67.8999999985098	2	3380	0	1
727.80000000447	606	38840	4491	921
76.8999999985098	2	3380	0	1
649.199999995529	587	38004	4440	899
66.7000000029802	2	3380	0	1

B Obsah elektronických dat

`kód/`
: Adresář s doprovodným kódem k práci.

`text/`
: Adresář s textem práce ve formátu PDF, včetně příloh, a všechny soubory potřebné pro vytvoření PDF dokumentu textu.

`README.txt`
: Textový soubor s informacemi o spuštění přiloženého kódu.

Literatura

[1] *The 12+ Best Web Analytics Tools to Improve Your Site*. [online]. [cit. 2023-12-30]. Dostupný z: ⟨https://www.hotjar.com/web-analytics/tools/⟩.

[2] Flanagan, David. *JavaScript: The definitive guide*. 7. vyd. 2020.

[3] Frisbie, Matt. *Professional JavaScript for Web Developers*. 5. vyd. 2023.

[4] Haverbeke, Marijn. *Eloquent JavaScript*. Třetí vyd. 2018.

[5] *ECMA-262 - Ecma International*. [online]. [cit. 2023-12-30]. Dostupný z: ⟨https://ecma-international.org/publications-and-standards/standards/ecma-262/⟩.

[6] Collins, Mark J. *Pro HTML5 with CSS, JavaScript, and Multimedia: Complete Website Development and Best Practices*. První vyd. 2017.

[7] *MutationObserver - Web APIs | MDN*. [online]. [cit. 2023-12-30]. Dostupný z: ⟨https://developer.mozilla.org/en-US/docs/Web/API/MutationObserver⟩.

[8] *Event reference | MDN*. [online]. [cit. 2023-12-30]. Dostupný z: ⟨https://developer.mozilla.org/en-US/docs/Web/Events⟩.

[9] *Promise - JavaScript | MDN*. [online]. [cit. 2023-12-30]. Dostupný z: ⟨https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise⟩.

[10] *Using Web Workers - Web APIs | MDN*. [online]. [cit. 2023-12-30]. Dostupný z: ⟨https://developer.mozilla.org/en-US/docs/Web/API/Web_Workers_API/Using_web_workers⟩.

[11] *The event loop - JavaScript | MDN*. [online]. [cit. 2023-12-30]. Dostupný z: ⟨https://developer.mozilla.org/en-US/docs/Web/JavaScript/Event_loop⟩.

[12] *Tasks, microtasks, queues and schedules - JakeArchibald.com*. [online]. [cit. 2023-12-30]. Dostupný z: ⟨https://jakearchibald.com/2015/tasks-microtasks-queues-and-schedules/⟩.

[13] *The JavaScript Event Loop*. [online]. [cit. 2023-12-30]. Dostupný z: ⟨https://flaviocopes.com/javascript-event-loop/⟩.

[14] *The event loop - JavaScript | MDN*. [online]. [cit. 2023-12-30]. Dostupný z: ⟨https://developer.mozilla.org/en-US/docs/Web/JavaScript/EventLoop⟩.

[15] *Window: requestAnimationFrame() method - Web APIs | MDN*. [online]. [cit. 2023-12-30]. Dostupný z: ⟨https://developer.mozilla.org/en-US/docs/Web/API/window/requestAnimationFrame⟩.

[16] *setTimeout() global function - Web APIs | MDN*. [online]. [cit. 2023-12-30]. Dostupný z: ⟨https://developer.mozilla.org/en-US/docs/Web/API/setTimeout⟩.

[17] *setInterval() global function - Web APIs | MDN*. [online]. [cit. 2023-12-30]. Dostupný z: ⟨https://developer.mozilla.org/en-US/docs/Web/API/setInterval⟩.

[18] *Monkey patch - Wikipedia*. [online]. [cit. 2023-12-30]. Dostupný z: ⟨https://en.wikipedia.org/wiki/Monkey_patch⟩.

[19] *Site Isolation*. [online]. [cit. 2023-12-30]. Dostupný z: ⟨https://www.chromium.org/Home/chromium-security/site-isolation/⟩.

[20] *Inside look at modern browsers*. [online]. [cit. 2023-12-30]. Dostupný z: ⟨https://developer.chrome.com/blog/inside-browser-part1⟩.

[21] *Sandbox*. [online]. [cit. 2023-12-30]. Dostupný z: ⟨https://chromium.googlesource.com/chromium/src/+/HEAD/docs/design/sandbox.md⟩.

[22] *Virtual DOM - Wikipedia*. [online]. [cit. 2023-12-30]. Dostupný z: ⟨https://en.wikipedia.org/wiki/Virtual_DOM⟩.

[23] *Virtual DOM and Internals - React*. [online]. [cit. 2023-12-30]. Dostupný z: ⟨https://reactjs.org/docs/faq-internals.html⟩.

[24] *React Virtual DOM Explained in Simple English - Programming with Mosh*. [online]. [cit. 2023-12-30]. Dostupný z: ⟨https://programmingwithmosh.com/react/react-virtual-dom-explained/⟩.

[25] *rrdom*. [online]. [cit. 2023-12-30]. Dostupný z: ⟨https://github.com/rrweb-io/rrweb/tree/master/packages/rrdom⟩.