Česká zemědělská univerzita v Praze

Provozně ekonomická fakulta

Katedra informačního inženýrství

Bakalářská práce

Alternativy jazyka JavaScript v prostředí webu na straně klienta

Martin Homza

ČESKÁ ZEMĚDĚLSKÁ UNIVERZITA V PRAZE

Provozně ekonomická fakulta

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Martin Homza

Systémové inženýrství a informatika
Informatika

Název práce

Alternativy jazyka JavaScript v prostředí webu na straně klienta

Název anglicky

JavaScript language alternatives in a web environment on client-side

---

Cíle práce

Cílem práce je najít, popsat a zhodnotit dostupné alternativy jazyka Javascript pomocí malé webové aplikace. Výsledem práce bude vícekriteriální vyhodnocení testovaných variant z pohledu uživatele i z pohledu tvůrce.

Metodika

Budou dodržovány standardy softwarového inženýrství, především UML. V obou testovaných pohledech bude zvoleno 5 až 7 testovaných parametrů, které budou vyhodnocovány na pětistupňové ordinální škále. Výsledné vícekriteriální srovnání bude také ve formě neúplně ordinálně uspořádané množiny.

Doporučený rozsah práce

60 – 80 stran

Klíčová slova

JavaScript; web; client-side; WebAssembly; TypeScript; Dart; CoffeeScript; Elm; Blazor

---

Doporučené zdroje informací

FENTON, Steve (2017) Pro typescript. New York, NY: Springer Science+Business Media, ISBN 9781484232484.

HIMSHOOT, Peter (2019) Blazor revealed – building web applications in .NET. New York, NY, Springer Science+Business Media, ISBN 9781484243428.

MACCAW, Alex (2012) The little book on CoffeeScript. Sebastopol, CA: O'Reilly Media, ISBN 9781449321055.

ŽÁRA, Ondřej (2015) JavaScript: Programátorské techniky a webové technologie, Brno Computer Press, ISBN 9788025145739.

Předběžný termín obhajoby

2020/21 LS – PEF

Vedoucí práce

doc. Ing. Vojtěch Merunka, Ph.D.

Garantující pracoviště

Katedra informačního inženýrství

Elektronicky schváleno dne 19. 11. 2020

Ing. Martin Pelikán, Ph.D.

Vedoucí katedry

Elektronicky schváleno dne 19. 11. 2020

Ing. Martin Pelikán, Ph.D.

Děkan

V Praze dne 04. 02. 2021

Čestné prohlášení

Prohlašuji, že svou bakalářskou práci "Alternativy jazyka JavaScript v prostředí webu na straně klienta" jsem vypracoval samostatně pod vedením vedoucího bakalářské práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou citovány v práci a uvedeny v seznamu použitých zdrojů na konci práce. Jako autor uvedené bakalářské práce dále prohlašuji, že jsem v souvislosti s jejím vytvořením neporušil autorská práva třetích osob.

V Praze dne 15.3.2021 _____

Poděkování

Rád bych touto cestou poděkoval doc. Ing. Vojtěchu Merunkovi, Ph.D. za vedení a pomoc při zpracování této bakalářské práce a také své rodině za podporu.

Alternativy jazyka JavaScript v prostředí webu na straně klienta

Abstrakt

Tato práce pojednává o porovnání významných dostupných alternativ programovacího jazyka JavaScript v prostředí webu na klientské straně. Hlavním cílem je porovnání programovacích jazyků TypeScript, CoffeeScript, Dart a frameworku Blazor s jazykem C# mezi sebou s ohledem na vývojáře a jejich cílené uživatele a zjištění výhod použití těchto jazyků namísto JavaScriptu.

Teoretická část práce rozebírá prostředí webové aplikace a technologie pro vývoj webových aplikací, jazyk JavaScript a důvody vzniku alternativních jazyků. Další část popisuje jednotlivé alternativní jazyky, jejich funkcionality, technologie a výhody proti JavaScriptu.

Praktická část obsahuje návrh a implementace ukázkových webových aplikací v jednotlivých alternativních jazycích, jejich popis a výsledné porovnání a zhodnocení.

Klíčová slova: JavaScript; web; klientská strana; WebAssembly; TypeScript; Dart; CoffeeScript; Elm; Blazor

JavaScript language alternatives in a web environment on client-side

Abstract

This thesis is devoted to a comparison of an important available alternatives of the JavaScript programming language in the web environment on the client-side. The main purpose is to compare the programming languages TypeScript, CoffeeScript, Dart and Blazor framework with the C# language regarding to developers and their targeted users and finding out the advantages of using these languages instead of JavaScript.

The theoretical part of this thesis analyses the environment of the web application and technologies for web applications development, JavaScript language and the reasons for the emergence of alternative languages. The next part describes individual alternative languages, their functionalities, technologies, and advantages against the JavaScript.

The practical part contains the design and implementation of the sample web applications in various alternative languages, their description and the resulting comparison and evaluation.

Keywords: JavaScript; web; client-side; WebAssembly; TypeScript; Dart; CoffeeScript; Elm; Blazor

Obsah

Seznam obrázků

Seznam tabulek

Seznam použitých zkratek

- HTML – Značkovací jazyk pro tvorbu webových stránek.
- SPA – Webová aplikace používající pouze jednu stránku pro všechen obsah.
- UML – Grafický jazyk pro návrh a dokumentaci programových systémů.
- WASM – Binární instrukční formát optimalizované pro webové prohlížeče.
- DOM – Objektově orientovaný model reprezentující XML, či HTML dokument.
- HTTP – Protokol pro komunikaci v počítačové síti.
- JIT – Metoda překladu zdrojového kódu programovacího jazyka využívající mezikód.
- API – Rozhrání pro programování aplikací.
- JSON – Formát zápisu dat.
- SQL – Strukturovaný dotazovací jazyk pro práci s daty v relačních databázích.
- CSS – Jazyk pro popis způsobu zobrazení HTML elementů na stránce.
- GUI – Grafické uživatelské rozhrání.
- SDK – Sada vývojových nástrojů pro vývoj aplikací.
- .NET – Softwarová platforma obsahující soubor technologií společnosti Microsoft

Úvod

Popularita webových aplikací a celkově i celého webu je čím dál vyšší. V současné době existuje již velké množství webových aplikací a mnoho aplikací určených původně pouze pro počítač, či pouze pro mobilní telefon byly převedeny na webové aplikace. Webové aplikace se těší úspěchu zejména díky jejich dostupnosti a jednoduchosti, neboť uživatel musí mít pro jejich používání pouze nainstalovaný libovolný webový prohlížeč. Díky současným technologiím lze s webovými aplikacemi pracovat stejně jako s těmi určenými pouze pro dané zařízení a nabízejí i podobný, či dokonce stejný výkon.

Pro vývoj webových aplikací je nutné používat poměrně velké množství technologií, neboť webová aplikace lze rozdělit do několika oblastí a každá tato oblast využívá jinou technologii. Významnou technologií je skriptovací jazyk JavaScript, dlouhou dobu dominantní technologie pro webové klientské zpracování dat. JavaScript se pyšní velké popularitě a jedná se o nejvíce používaný jazyk ze všech. Jazyk má však mnoho nevýhod a odlišností od ostatních populárních standardních jazyků, což vede k častému zmatení vývojářů, a tedy i k častým problémům s vývojem a během programu.

JavaScript byl dlouhou dobu jediným podporovaným jazykem prohlížeči pro klientské skriptování. Jeho problematické konstrukce a nedostupnost jiných možností pro klientské skriptování vedli vývojáře, a i celé společnosti k myšlence nahradit JavaScript jiným jazykem, či vytvořit alespoň další alternativní jazyk. Tyto snahy o nahrazení jazyka JavaScript jej provázeli celým jeho historickým vývojem, nakonec však spousta alternativ zanikla a žádné se nepovedlo JavaScript zcela nahradit. V současné době však existují významné a oblíbené alternativy, a některé dokonce oblíbeností i dohánějí samotný JavaScript. Situace alternativ jazyka JavaScript v prostředí klientského zpracování dat se také významně zlepšila vznikem nového standardu WebAssembly, umožňující pro klientské zpracování dat využívat libovolný programovací jazyk.

Právě těmito alternativami se zabývá tato práce, konkrétně budou vybrány významné jazyky CoffeeScript, Dart, TypeScript a framework Blazor, který využívá zmíněný standard WebAssembly. V teoretické části práce bude nejprve představeno samotné prostředí webu a jeho vývoj. Dále budou představeny jednotlivé části webové aplikace, či stránky z pohledu jejich tvorby a konkrétní technologie, které se při implementaci používají. Po představení webu a webových aplikací bude možné popsat přímo jazyk JavaScript a důvody vzniku jeho alternativ. Zbytek teoretické části bude věnován popisu jednotlivých zmíněných alternativ, konkrétně budou rozebrány jejich obecné informace, historický vývoj, výhody a nevýhody a styl zápisu.

Praktická část bude věnována hlavnímu cíli práce, tedy porovnání vybraných alternativ jazyka JavaScript mezi sebou a jejich zhodnocení. Pro získání výsledků, ze kterých následné porovnání alternativ bude vycházet, bude v každé alternativě vytvořena a zkoumána totožná demonstrační malá webová aplikace. Postupný vývoj aplikace bude v praktické části popsán a bude se držet standardů softwarového inženýrství. Zjištěné výsledky z aplikací budou porovnány a zhodnoceny jak z pohledu vývojáře, tak i z pohledu uživatele. Výsledky budou zobrazeny na neúplně ordinálně uspořádané množině.

Cíl práce a metodika

1.1 Cíl práce

Cílem práce je najít, popsat a zhodnotit dostupné alternativy jazyka Javascript pomocí malé webové aplikace. Výsledem práce bude vícekriteriální vyhodnocení testovaných variant z pohledu uživatele i z pohledu tvůrce.

1.2 Metodika

Budou dodržovány standardy softwarového inženýrství, především UML. V obou testovaných pohledech bude zvoleno 5 až 7 testovaných parametrů, které budou vyhodnocovány na pětistupňové ordinální škále. Výsledné vícekriteriální srovnání bude také ve formě neúplně ordinálně uspořádané množiny.

Teoretická východiska

1.3 Prostředí webu

Kapitola popisuje historický vývoj webového prostředí a vymezuje základní potřebné pojmy pro tvorbu webových aplikací a stránek. Jsou zde rozebrány základní druhy obsahu v prostředí webu, prohlížeče a Document Object Model.

1.3.1 Historické pozadí vývoje webu

I přes to, že za počátek internetu se považuje již rok 1969, samotný web vznikl až v roce 1989, když Tim Berners-Lee definoval hypertextový systém, nato vytvořil i hypertextový editor a první webový server, na kterém publikoval specifikaci standardů UDI (nyní URI), HTML a HTTP (Tim Berners-Lee, 1998).

Za skutečný počátek webu se však považuje až rok 1991, když byl zpřístupněn širší veřejnosti díky prvním grafickým webovém prohlížečům. Roku 1993 Marc Andreessen a Eric Bina vydali významný prohlížeč Mosaic. Mosaic dokázal zobrazovat pouze statické stránky, dokázal tedy pouze zpracovávat „DOM“ (Document Object Model). Marc Andreessen o rok později založil společnost „Netscape“ a vydal konkurenční populární prohlížeč „Navigator“. Andreessen měl vizi o prohlížečích zobrazujících dynamické webové stránky, tedy stránky, které dokážou interagovat s uživatelem, automatizovat operace a podobně. Proto najal Brendana Eicha, který vytvořil jazyk JavaScript (Fireship, 2019).

Ve stejné době společně s JavaScriptem vznikal i další důležitý jazyk pro rozvoj webu, konkrétně jazyk PHP pro serverové skriptování. Microsoft po vydání JavaScriptu vytvořil svou vlastní verzi interpretu JavaScriptu pro svůj prohlížeč Internet Explorer. Interpret Microsoftu měl mnoho odlišností od původního interpretu, a tak vznikaly problémy s kompatibilitou. V roce 1996 byl vydán jazyk CSS a HTML byl doplněn o podstatná rozšíření. Ke konci 20. století ovládal Internet Explorer 90 % trhu (Champeon, 2001).

Na začátku 21. století začala i komunita rozvíjet jazyk JavaScript knihovnami (AJAX, jQuery, …) a taktéž se začaly objevovat první snahy o nahrazení JavaScriptu, v té době jediného jazyku pro klientské skriptování. Tehdy se o jeho nahrazení pokoušel jazyk ActionScript. Internet Explorer se vznikem webového prohlížeče Firefox začal přicházet o dominantní postavení na trhu až nakonec se tohoto dominantního vedení ujal webový prohlížeč Chrome od společnosti Google. Chrome byl úspěšný předně díky své rychlosti, o kterou se zasloužil implementací modulu JIT (just-in-time compilation) do svého interpretu JavaScriptu „V8“. Po roce 2010 začaly vznikat populární JavaScriptové frameworky (Angular, Backbone, …), díky kterým se začaly používat JavaScriptové frameworky jako standard. Současně však vznikaly také další jazyky, se snahou nahradit JavaScript. Například CoffeeScript se snažil zakrýt „zvláštní“ funkcionality JavaScriptu a zkrátit jeho zápis. CoffeeScript však nad JavaScriptem nevyhrál, a tak se při používání musí překládat do JavaScriptu. Na druhou stranu se tím zavedlo používání transpilace jako běžný standard. Taktéž samotný Google vytvářel vlastní náhradu JavaScriptu, jazyk „Dart“, ten však také skončil jako CoffeeScript. Microsoft vydal také svůj jazyk, avšak nesnažil se JavaScript nahradit, ale doplnit je o potřebné funkcionality, které jsou nutné pro

tvorbu velkých projektů. TypeScript je nyní nejvíce používány jazyk jako alternativa JavaScriptu. (Fireship, 2019)

V roce 2017 začal být v nejpoužívanějších prohlížečích podporován webový standard WebAssembly popisující strojový kód pro prohlížeče. Což umožnilo kód jakéhokoliv programovacího jazyku spouštět v prohlížeči. Již tedy není nutné používat JavaScript a lze používat libovolný jazyk s překladačem do formátu WebAssembly (Drozdík, 2019).

1.3.2 Webové prohlížeče

Webový prohlížeč je aplikace zpracovávající a zobrazující obsah webových stránek. Webových prohlížečů je velké množství a zejména dříve se často lišily funkcionalitami, podporou technologií, vzhledem, či způsobem zpracování i výstupem stránek. Z těchto důvodů je potřeba používat takové technologie, aby vytvářená aplikace byla podporována co největším množstvím různých prohlížečů (DifferenceBetween.net, 2011). Nekompatibilita technologií v prohlížečích byla způsobena hlavně malou mírou standardizace technologií a rychlým rozmachem dynamických webových stránek bez dostatečných rozhraní.

Mezi nejznámější prohlížeče současnosti patří Microsoft Edge, Internet Explorer, Chrome, Firefox, Brave, Opera a Safari (Žára, 2015, s. 23-24).

1.3.3 Statické webové stránky

Prvním obsahem sdíleným v prostředí webu byly statické webové stránky. Web měl původně sloužit ke sdílení dokumentů, a tak jsou statické webové stránky v podstatě obyčejnými dokumenty s textem a jednoduchým formátováním. Statické webové stránky až na funkci pro přesměrovávání mezi stránkami postrádají interaktivitu (Niemczyk, 2015, s.12).

Většinou uživatel získává statické stránky do prohlížeče ve stejném stavu jako jsou uloženy na serveru. Jelikož zobrazují vždy stejný obsah, nelze je odlišit podle příjemce, ani jiných parametrů. Z pohledu implementace se může jednat o HTML dokumenty uložené v nějakém souborovém systému.

V dnešní době statické stránky využívají i CSS a nějaký skriptovací jazyk na klientské straně. Ovšem nepoužívají žádná data ze serverů a serverových databází, a tak je obsah stále stejný. Používají se pro stránky, které není potřeba aktualizovat.

Využívání statických stránek namísto dynamických stále přináší výhody jako je bezpečnost, vysoká rychlost, nezávislost na databázích a aplikačních serverech a jednoduchost (Lamolony, 2019).

1.3.4 Dynamické webové stránky

Kvůli nemožnosti interaktivity bylo nutno přejít ze statických webových stránek na webové stránky dynamické. Interaktivita je zde vytvářena pomocí programovacích jazyků na klientské a serverové straně. Stránky jsou generovány dle potřeby na serveru, či případně dotvářeny za běhu. Dynamické stránky umožnily existenci internetových obchodů, webových informačních systémů, webových her, či online novinám (Niemczyk, 2015, s.12).

V současnosti je většina webových stránek dynamických. Pro jejich implementaci je využíváno dvou způsobu. V prvním případě jde o stránky, kde samotný vývojář přímo spravuje serverovou část, naprogramovanou například v jazyce PHP, nebo ASP.NET a databázi. Ve druhém způsobu využívá vývojář pro správu stránek CMS neboli redakční systémy, jedná se například o WordPress, či Drupal.

Dynamické stránky zobrazují obsah uživatele v HTML šabloně a umožňují tento obsah editovat. Pokud uživatel provede změny obsah mu je znovu vygenerován. Hlavní výhody dynamických webových stránek je jednoduchá aktualizace obsahu, zejména při používání redakčního systému a vysoká flexibilita, umožňující vytvářet různé interaktivní funkcionality (Lamolony, 2019).

1.3.5 Webová aplikace

Nyní web umožňuje vytvářet stejné aplikace jako jsou na počítačích, či mobilních telefonech. Nejedná se o pouhé stránky zobrazující obsah ale o aplikace, které zpracovávají uživatelská data, komunikují se serverem a pracují s databázemi. Jejich výhodou proti normálním aplikacím je větší dostupnost (Procházka, 2019, s. 12).

Na rozdíl od dynamických webových stránek, kde převážné množství zaslaných dat směřuje ze serveru k uživateli, webové aplikace využívají plně obousměrnou komunikaci mezi klientem a serverem. Webové aplikace tak umožňují zpracovávat specifická dynamická data uživatele, s těmito daty je možné pracovat skrz aplikaci, publikovat je, přeposílat mezi uživatele a další.

Mezi webové aplikace patří například webové hry, textové editory, editory fotografií a aplikace pro obchodování s akciemi (Shukla, 2017).

1.3.6 Document Object Model

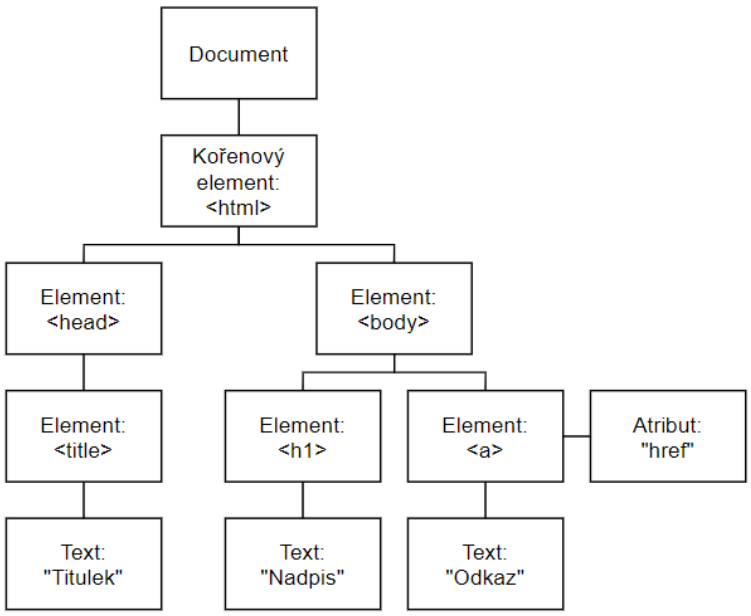
Document Object Model (DOM) je velmi podstatné rozhrání pro vývoj webových stránek a aplikací, neboť jej každá stránka musí obsahovat, aby bylo možné propojit tuto stránku se skripty, či programovacími jazyky. Jedná se o objektově orientované rozhrání, nezávislé na platformě, reprezentující strukturu dokumentu jako logický strom. Tento logický strom obsahuje větve zakončené uzly. Každý uzel obsahuje objekty, tvořící strukturu dokumentu. Díky tomuto rozhrání je možné pomocí skriptu měnit obsah dokumentu, formátování a strukturu. Uzly také dokážou detekovat události, na které může skript reagovat (MDN Web Docs, 2021).

Při vykreslování obsahu prohlížečem musí kromě vykreslování elementů, také vypočítávat kam je vykreslovat podle velikosti stránky. Kdykoliv se změní velikost jednoho elementu, prohlížeč musí upravit všechny elementy na stránce. Kvůli tomu je DOM výpočetně náročný a ve výsledku i pomalý. Pro menší zatěžování modelu při aktualizaci vznikl virtuální DOM a inkrementální DOM. Tyto techniky používají například JavaScriptové frameworky React a Angluar, nebo framework pro jazyk C#, Blazor.

Virtuální DOM umožňuje vytvářet a zpracovávat v programu virtuální objekty reprezentující DOM. Zpracování těchto virtuálních objektů v programu je mnohem rychlejší než ve skutečném DOM. Při potřebě vykreslení tohoto objektu ve skutečném DOM se v něm pouze udělají takové změny, aby vypadal jako ten virtuální.

Inkrementální DOM je technika minimalizují množství změn potřebných pro aktualizaci DOM do výsledné podoby (Blazor University, 2021).

Na obrázku 1 je vykreslena schématická struktura DOM modelu.



Obrázek 1 – Model DOM (W3Schools, c1999-2021)

1.4 Implementace webových stránek a aplikací

Při procesu implementace webových stránek a aplikací je potřeba zaměřit se na několik různých částí, které lze rozdělit podle použitých technologií a oblastí a do kterých lze každou webovou stránku a aplikaci rozdělit. Některé tyto části umožňují vývojáři pro implementaci vybrat si libovolnou technologii z více variant. Ostatní části naopak vyžadují využití jedné dané technologie pro vývoj, i přes to však existují alternativy. Tato kapitola rozebírá tyto základní části webové stránky od struktury a grafiky po zpracování dat na serveru a ukládání těchto dat. Dále vysvětluje proces transpilace a standard WebAssembly.

1.4.1 Struktura a obsah

Pro tvorbu struktury a obsahu je na webu dominantně využíván značkovací jazyk HTML (Hypertext Markup Language), navržený pro zobrazování v prohlížečích s kořeny v jazyce SGML. Jazyk původně vznikl pro formátování a šíření vědeckých dokumentů, v současné době se však používá pro všechny typy dokumentů na webu a umožňuje využívat pokročilé vizuální funkce. Významným milníkem pro HTML je jeho pátá verze, která z jednoduchého značkovacího jazyka pro statické webové stránky vytvořila plnohodnotné rozhrání API. Pátá verze aktualizovala elementy z předchozích verzí, přidala nové elementy, podporu multimédií a rozhrání API pro model DOM. V dokumentu lze také používat a připojovat CSS stránky a JavaScriptové skripty, což vypovídá o provázanosti HTML s těmito technologiemi.

Jazyk HTML má hierarchickou strukturu. Jeho základním stavebním kamenem jsou elementy pocházející z množiny definovaných značek, které mají sémantický význam a jsou reprezentované tagy neboli značkami. Element obsahuje počáteční značku a musí obsahovat i ukončovací, pokud uchovává nějaký obsah. Značka nese název elementu a může obsahovat i atributy, které jsou složené z názvu a hodnoty. Struktura dokumentu musí mít povinné části HTML, tedy deklaraci verze jazyka, kořenový element, hlavičku pro připojení externích dokumentů a pro technické informace a tělo pro zobrazovaný obsah (Brown, 2014, s. 13-31).

Tento jazyk je jediným podporovaným jazykem v prohlížečích pro tvorbu struktury webové stránky. Jako alternativu lze použít různé nástroje, například takzvané „templating engine“, které zjednodušují zápis HTML a přidávají do něj funkcionality z programovacích jazyků, tedy podmínky, cykly, funkce a další. Zástupci těchto nástrojů jsou Pug.js a Razor (Hiwarale, 2018).

V tabulce 1 je ukázán rozdílný zápis jazyka HTML a nástroje Pug.js.

Tabulka 1 – Porovnání jazyka HTML a nástroje Pug.js (Vlastní zpracování)

HTML	Pug.js

`<html lang="cs">`
`  <head>`
`  <title>Titulek</title>`
`  </head>`
`  <body>`
`  <h1>Nadpis</h1>`
`  <a href="#">Odkaz</a>`
`  </body>`
`</html>`
	`doctype html`
	`html(lang='cs')`
	`  head`
	`  title Titulek`
	`  body`
	`  h1 Nadpis`
	`  a(href='#') Odkaz`

1.4.2 Grafika

Tvorbu grafiky umožňuje na webu jazyk CSS (Cascading Style Sheets), česky kaskádové styly. Vznikl za účelem oddělení struktury od grafického zobrazení v HTML dokumentu, zastává tedy nejčastěji roli doplňku k HTML. Popisuje způsob zobrazení elementů, tedy upravuje vzhled obsahu, barvy, uspořádání písma, uspořádání dle velikosti zařízení, animace a další. Aktuálně má jazyk nejnovější třetí verzi.

Syntaxe jazyka obsahuje pouze selektory, vlastnosti a hodnoty vlastností. Selektor označuje elementy v dokumentu podle názvu, atributu, či polohy. Selektor dále obsahuje již zmíněné vlastnosti a hodnoty, které umožňují formátovat elementy (Lazaris, 2014, s.15-22).

Přestože CSS kód má jednoduchý zápis, se zvětšujícím se obsahem kódu je čím dál komplikovanější jeho správa. Proto se často, zejména při velkých projektech používají v CSS různá metodika. Mezi nejpoužívanější metodika patří OOCSS, členící styly do objektů, BEM a SUIT CSS, určující způsob pojmenovávání tříd podle struktury a ITCSS, které rozděluje CSS soubor do více souborů podle jejich účelu a řadí je podle specifičnosti, velikosti obsahu a důsledku při změně (Michálek, 2020).

CSS je opět jediným podporovaným jazykem pro formátování HTML v prohlížeči. Jako alternativu lze využívat CSS preprocesory. Jedná se o jazyky, které jsou nadmnožinou jazyka CSS, přidávají do něj nové vlastnosti jako vnořování selektorů, jednoduší zápis, podmínky, cykly a jejich výstup se překládá do samotného CSS. Nejznámějšími preprocesory jsou LESS, SASS a Stylus (Michálek, 2014).

Tabulka 2 - Porovnání jazyka CSS a preprocesoru SASS (Vlastní zpracování)

CSS	SASS

`.Button {`
`  width: 30px;`
`  background-color: blue;`
`}`
`.Button:hover {`
`  backgroud-color: cyan;`
`}`
	`$buttonWidth: 30px;`
	
	`.Button {`
	`  width: $buttonWidth;`
	`  background-color: blue;`
	`  &:hover {`
	`  backgroud-color: cyan;`
	`  }`
	`}`

1.4.3 Zpracování dat na klientské straně

Již zmíněné dynamické webové stránky měly, zejména u větších projektů nevýhodu v tom, že všechna data byla zpracovávána na serverové straně. To vedlo jednak ke značné zátěži na servery, ale také k objemné datové komunikaci mezi klientem a serverem. Proto byla potřeba část zpracování dat přenechat přímo prohlížeči uživatele. Díky tomu se více prosadil jazyk JavaScript, který jako první vznikl, aby umožnil dynamičnost webových stránek a byl tak po dlouhou dobu jediným podporovaným jazykem v prohlížečích. Tento vývoj způsobil přechod z plně dynamických stránek na více statické stránky, kde se načte celá stránka do prohlížeče pouze jednou a další data jsou načítány postupně dle potřeb uživatele. Tím však vzrostly nároky na JavaScript, což odhalilo jeho dosavadní omezení a bylo potřeba jej tak dál rozvíjet (Niemczyk, 2015, s.13).

Aby se vyřešily problémy a omezení JavaScriptu, začaly vznikat kromě knihoven a frameworků také jeho alternativy. Tyto alternativy nabízely možnost používat upravenou

verzi JavaScriptu, které přidávaly některé funkcionality jako například typové anotace, či naprosto jiný programovací jazyk. Některé alternativy byly vyvinuty pouze jako další možnost pro skriptování na klientské straně, jiné byly vytvořeny s cílem JavaScript plně nahradit. Jelikož však JavaScript byl jediným podporovaným jazykem v prohlížečích musely se zdrojové kódy napsané v těchto alternativních programovacích jazycích transpilovat do JavaScriptu. Mezi významné alternativy JavaScriptu patří TypeScript, CoffeeScript, Dart a Elm (Žára, 2015, s. 28).

Modernějším způsobem zpracování dat na klientské straně je využití standardu WebAssembly (Himshoot, 2019, s. xvii).

1.4.3.1 Transpilace

Pojem transpilace je často zaměňován s pojmem kompilace. Kompilace popisuje proces konverze zdrojového kódu napsaného v nějakém programovacím jazyce do jiného programovacího jazyka. Transpilace je specifickým typem kompilace, popisujícím proces konverze zdrojového kódu do jiného programovacího jazyka, který má podobnou úroveň abstrakce jako programovací kód, ve kterém je původní zdrojový kód napsán. Proces konverze zdrojového kódu vysokoúrovňového jazyka do assembleru se tedy nazývá kompilace a proces konverze zdrojového kódu TypeScriptu do JavaScriptu se nazývá transpilace, neboť mají podobnou úroveň abstrakce. Další příklady transpilace mohou být převody zdrojových kódu z jazyka C++ do jazyka C, z jazyka CoffeeScript do jazyka JavaScript, nebo převod z jazyka PHP to jazyka C++ (Fenton, 2017, s. xxii).

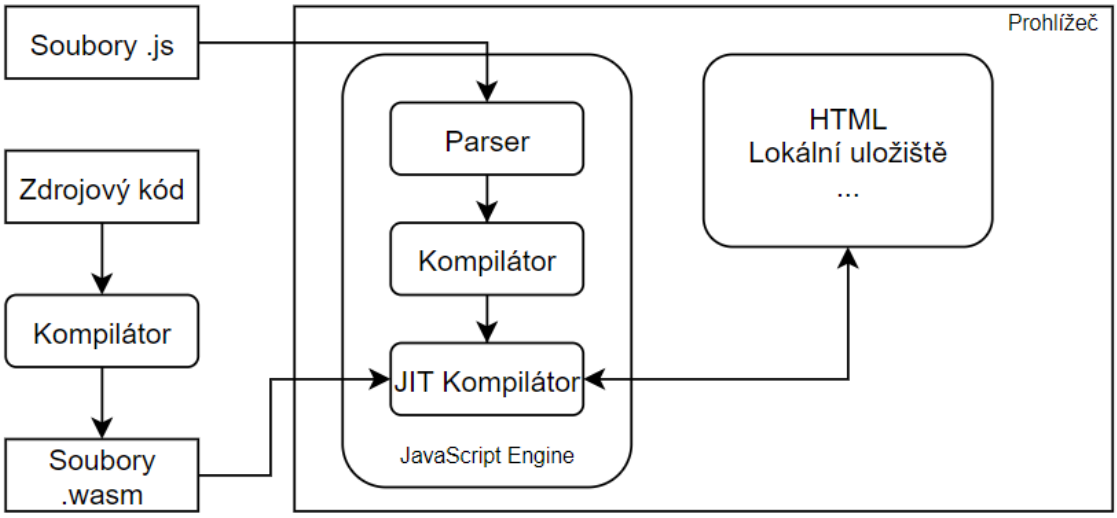
V tabulce 3 je ukázán rozdíl kompilace a transpilace na jednoduché funkci vytvořené v jazyce C.

Tabulka 3 - Porovnání kompilace a transpilace (Vlastní zpracování)

C	JavaScript	Assembler
`int add(int num1, int num2) { return num1 + num2; }`	`function add(num1, num2) { return num1 + num2; }`	`add(int, int): push rbp mov rbp, rsp mov DWORD PTR [rbp-4], edi mov DWORD PTR [rbp-8], esi mov edx, DWORD PTR [rbp-4] mov eax, DWORD PTR [rbp-8] add eax, edx pop rbp ret`

1.4.3.2 WebAssembly

WebAssembly (zkráceně Wasm) je standard navržený organizací W3C, definující nový binární instrukční formát optimalizovaný pro web. Je vyvinut jako výstup kompilace vysokoúrovňových programovacích jazyků, který umožňuje běh na webu. Z principiálního pohledu WebAssembly popisuje dva ekvivalentní formáty. První formát je Bytecode, podobný assembleru a druhý je Binarycode podobný strojovému kódu (Drozdík, 2019). Tedy WebAssembly umožňuje zkompilovat zdrojový kód vytvořený v libovolném vysokoúrovňovém programovacím jazyce do formátu zvaného WASM. Zdrojový kód v tomto formátu je pak při spuštění stažen prohlížečem, který jej pomocí jednotky JIT zkompiluje do nativního kódu prohlížeče a může jej spouštět (Himshoot, 2019, s. xvii).



Obrázek 2 – Proces zpracování JavaScriptového kódu a kódu jiného jazyka ve formátu WASM (Himshoot, 2019, s. xviii)

WebAssembly přináší pro vývoj na webu velké množství výhod. Hlavní výhodou je nepochybně možnost využít libovolný programovací jazyk, který má kompilátor do formátu WASM. Další velkou výhodou je mnohem vyšší teoretická rychlost zpracování kódu oproti JavaScriptu, neboť se jedná o zkompilovaný kód v assembleru na rozdíl od JavaScriptu, který se v prohlížeči interpretuje. Významnou výhodou je také možnost využívat ve WebAssembly JavaScriptový kód, díky čemu může kód ve WebAssembly využívat JavaScriptové API a knihovny. Na druhou stranu současný stav WebAssembly přináší i nevýhody oproti JavaScriptu, které jsou zapříčiněné hlavně kratší dobou vývoje na rozdíl do JavaScriptu. Mezi tyto nevýhody patří absence spousty rozhrání API, kvůli čemu nemůže WebAssembly napřímo přistupovat k modelu DOM a musí k jeho úpravám využívat JavaScript. WebAssembly také postrádá mnoho funkcionalit jako jsou textové řetězce, či „Garbage Collector“, proces pro správu paměti. Nicméně některé programovací jazyky tyto funkcionality obsahují (Drozdík, 2019).

V současné době existují kompilátory do formátu WASM pro jazyky C++, Rust, C# a další (Himshoot, 2019, s. xix).

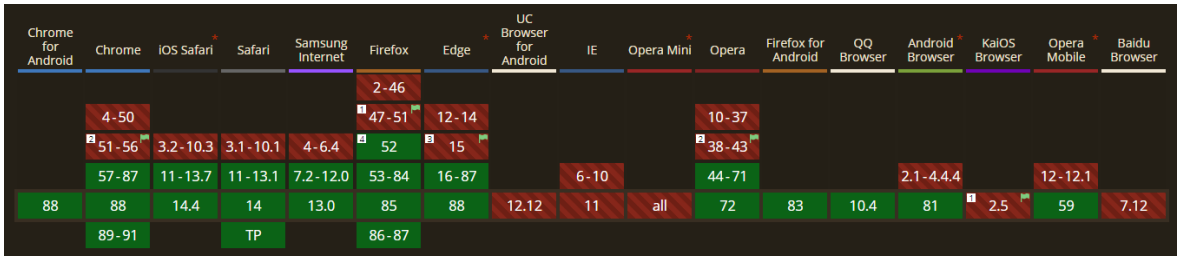
Tabulka 4 ukazuje výstup kompilace jednoduché funkce v jazyce C do formátu WASM prohlížeče Firefox.

Tabulka 4 - Kompilace do formátu WASM (Vlastní zpracování)

C	Firefox x86 Assembly

`int add(int num1, int num2) {`
`  return num1 + num2;`
`}`
	`wasm-function[0]:`
	`  sub rsp, 8`
	`  mov ecx, esi`
	`  mov eax, ecx`
	`  add eax, edi`
	`  nop`
	`  add rsp, 8`
	`  ret`

Formát WASM je podporován ve všech hlavních webových prohlížečích, tedy Chrome, Edge, Safari, Firefox, včetně jejich mobilních verzí. Například Internet Explorer však formát WASM podporovat nikdy nebude (Himshoot, 2019, s. xix). Na následujícím obrázku je možné vidět, které prohlížeče podporují formát WASM (zelené obdélníky) a které ho nepodporují (červené obdélníky). V současné době formát WASM podporuje přibližně 92 % procent nejpoužívanějších prohlížečů (Caniuse, nedatováno).



Obrázek 3 – Podpora formátu WASM v jednotlivých prohlížečích (Caniuse, nedatováno)

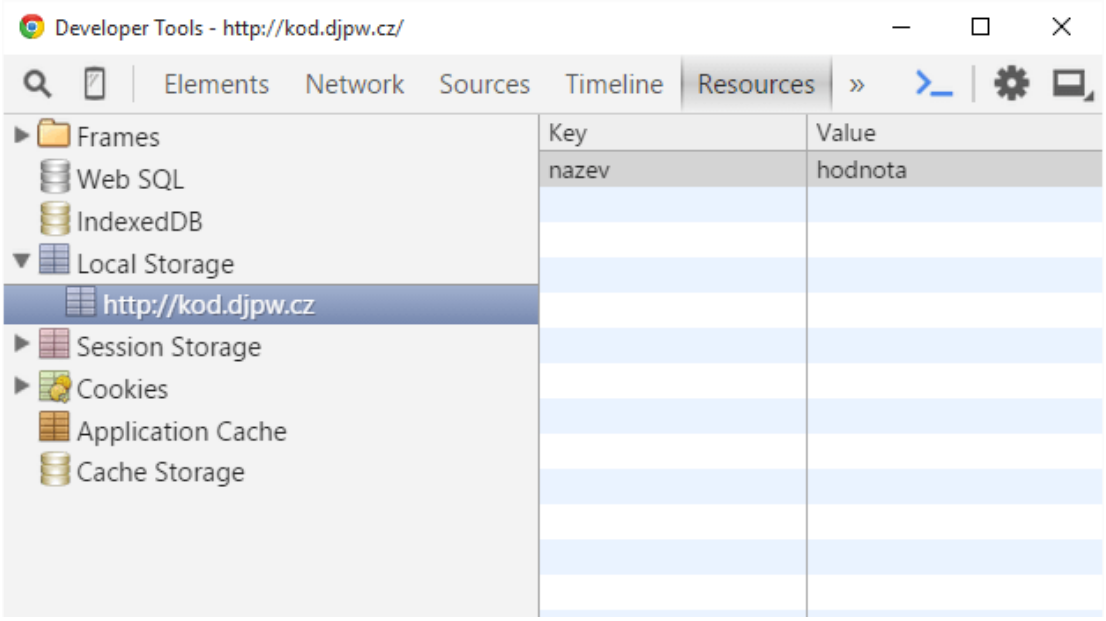
1.4.4 Zpracování dat na serverové straně

Zpracování dat na serverové straně je nezbytné pro dynamické webové stránky. Díky tomuto zpracování lze generovat a zpracovávat obsah konkrétního uživatele, či publikovat data z databáze. Také umožňuje různé interaktivní funkcionality jako zasílání notifikací, aktualizace pomocí emailu a další. Serverová strana komunikuje s prohlížečem pomocí HTTP protokolu a využívá GET a POST metody pro zasílání dat. Toto zpracování je implementováno pomocí nějakého skriptovacího jazyka. Zde není vývojář omezen tak jako na klientské straně a může zde využít libovolný programovací jazyk závislí pouze na použitém HTTP serveru. Mezi zástupce těchto jazyků patří PHP a ASP.NET (MDN Web Docs, 2021).

1.4.5 Ukládání dat

Pro ukládání dat na klientské straně je nutné používat dostupné technologie prohlížeče. Konkrétně lze pro ukládání dat na klientské straně využít uložiště „Session Storage“, „Local Storage“, „IndexedDB“, a „Cookies“. Každá tato varianta využívá jiné rozhrání API, má jinou dobu životnosti, omezení a možnosti.

- **Session Storage** – využívá stejně jako Local Storage rozhrání API „Web Storage“ a tedy obě tato uložiště ukládají data ve tvaru páru klíč/hodnota, také mají stejná pravidla pro užívání, liší se však v době životnosti. Toto uložiště má omezenou životnost pouze po dobu, dokud má uživatel spuštěnou stránku v prohlížeči, při zavření stránky se uložiště smaže. Pokud uživatel spustí tuto stránku současně v další záložce prohlížeče vytvoří se v této nové záložce nové uložiště nezávislé na prvním uložišti.
- **Local Storage** – toto uložiště má neomezenou dobu životnosti. Taktéž může více puštěných stránek současně přistupovat ke stejnému uložišti. Využívá se například pro ukládání dat se záměrem snížit objem přenosu dat po síti, pro ukládání dat, když uživatel nemá internetové připojení, nebo pro data, které není nutné přenášet.
- **IndexedDB** – umožňuje ukládat větší množství dat ve strukturované podobě, s rychlým vyhledáváním pomocí indexů. Pracuje také na rozdíl od prvních dvou uložišť asynchronně, aby neblokoval běh stránky.
- **Cookies** – první způsob ukládání dat na klientské straně, dnes je doporučováno používat namísto cookies předchozí zmíněná uložiště (Fenton, 2017, s. 174-176).



Obrázek 4 – Uložiště prohlížeče Chrome ve vývojářských nástrojích (Jahoda, 2016)

Výběr uložiště pro serverovou stranu nese stejnou volnost jako výběr skriptovacího jazyku pro serverové zpracování dat. Vývojář si tedy může vybrat libovolnou SQL databázi, NoSQL databázi, nebo může ukládat data do souborů.

1.5 JavaScript

JavaScript je prototypově objektově orientovaný skriptovací jazyk optimalizovaný pro běh v prohlížečích. Je primárně určený pro dynamické weby a zpracování dat na klientské straně. Standardizovanou verzí jazyka je „ECMAScript“. JavaScript se stal rychle součástí všech prohlížečů a dnes zastává místo mezi nejrozšířenějšími technologiemi. Název jazyka budí dojem, že se jedná o typ jazyka Java, určený pro klientské skriptování, podobnost názvu má však naznačovat podobnost zápisu kódu. Syntaxe tedy vychází z jazyka C a některá rozhraní a funkcionality z jazyka Java. Objektový a funkcionální přístup JavaScriptu vychází z jazyků Scheme a Self. Aby se jazyk odlišil od Javy, používá prototypovou dědičnost namísto třídní dědičnosti. JavaScript je dynamicky typovaný, datové typy se tedy přiřazují až za běhu programu. JavaScript může být zapisován přímo do HTML souboru vně elementu „script“ anebo do samostatného souboru s příponou „.js“ (Žára, 2015, s. 11-12).

1.5.1 Historický vývoj

Vývoj jazyka začal najmutím Brendana Eicha společností Netscape s úmyslem, aby pro jejich prohlížeč vytvořil skriptovací jazyk, díky kterému budou stránky více dynamické a budou schopné interakce s uživatelem. První verze jazyka byla navržena v roce 1995 během deseti dnů s názvem Mocha. Krátce poté byl přejmenován na LiveScript a se zavedením Java appletů do prohlížeče Navigator 2 dostal své současné jméno JavaScript.

Roku 1996 Microsoft vytvořil vlastní implementaci interpretu JavaScriptu s názvem JScript pro svůj prohlížeč Internet Explorer 3. Interpret Microsoftu měl odlišnosti od původního interpretu, což vedlo k nekompatibilitě a bylo tedy nutné standardizovat jazyk. Formální standardizaci spravuje organizace ECMA se standardem ECMAScript. Na konci devadesátých let byl vydána významná třetí verze standardu ECMA, která byla po delší dobu poslední verzí.

Roku 2008 vydala společnost Google svůj prohlížeč Chrome s modulem JIT, díky kterému byla jejich implementace interpretu zvaná „V8“ mnohem výkonnější oproti konkurenci. Díky tomu, že Google ukázal, jak může být vykonávání JavaScriptu výkonné, vznikla myšlenka využívat JavaScript i pro zpracování dat na serverové straně. V roce 2009 tak s cílem standardizovat serverová rozhraní pro JavaScriptu vznikla organizace CommonJS. Později téhož roku byl vytvořen projekt Node.js, obsahující rozhraní a knihovny pro interpret V8 umožňující spouštět jednoduše JavaScript na serverové straně. Projekt Node.js zastínil organizaci CommonJS a nyní je standardem pro zpracování dat na serverové straně pomocí JavaScriptu.

Posledními významnými verzemi ECMAScriptu je pátá verze obsahující striktní režim a funkce pro prací s formátem JSON a šestá verze, která přinesla zejména třídy, „Arrow funkce“ a nová klíčová slova pro konstanty a proměnné (Žára, 2015, s. 12-13).

1.5.2 Výhody jazyka

Jako každý programovací jazyk má i JavaScript své výhody a nevýhody. Řada problémů jazyka je způsobena tím, že se při jeho vývoji neočekávala tak velká rozšířenost, jakou má dnes. Nevýhody, nedostatky a odlišnosti jazyka byly také jistě podněty, jejichž důsledkem byl vývoj alternativních jazyků. JavaScript má však v určitých oblastech a za určitých

podmínek stále své výhody, které mohou zastínit výhody alternativních jazyků. Mezi největší výhody patří:

- **Rozšířenost** – má velkou komunitu, která pro něj neustále vyvíjí obsah v podobě knihoven, rozhraní a frameworků. Významným zástupcem knihoven je „jQuery“, zajišťující kompatibilitu jazyka mezi prohlížeči a zjednodušující práci s modelem DOM. JavaScriptové frameworky se vyvíjí nejvíce v posledních letech a značně zjednodušují tvorbu webových aplikací, jedná se například o „Angular“, „React“ a „Vue“.
- **Dostupnost pro různé platformy** – může být použit pro zpracování na serveru i u klienta v prohlížeči, kde je navíc provázaný s jazyky HTML a CSS (Žára, 2015).
- **Výkon** – může být u transpilovaných alternativ nižší z důvodu jiného zápisu a u formátu WASM kvůli chybějícímu rozhraní pro model DOM a méně pokročilému překladači (Drozdík, 2019).
- **Jednoduchost** – jazyk má jednoduchou syntaxi a snadný je i proces implementace a publikování aplikace (Žára, 2015).

1.5.3 Nevýhody a odlišnosti jazyka

„Na oblíbeném programátorském webu *stackoverflow.com* je položeno přes tři čtvrtě milionu otázek na téma JavaScript. To je dobrým dokladem zájmu i nedostatečného vzdělání mezi webovými vývojáři.“ (Žára, 2015, s. 9)

Časté používání alternativ vývojáři namísto JavaScriptu je pravděpodobně zapříčiněno jednak jeho nevýhodami, ale i jeho odlišnostmi od ostatních standardních jazyků. Mezi nevýhody jazyka patří:

- **Struktura** – jazyk se vyznačuje špatnou strukturou a nevhodností pro velké projekty
- **Nekompatibilita** – mezi prohlížeči se stále vyskytuje občasná nekompatibilita
- **Nedostatek standardních knihoven** – nutí vývojáře používat knihovny a frameworky třetích stran.
- **Složitá správa modulů** – způsobuje problémy se závislostmi. Problémy vznikají například, když jsou skripty přidány na stránku ve špatném pořadí, nebo pokud vývojář zapomene přidat značku pro nový skript do stránky, či v případě, že stránka obsahuje skript, který není používán. Podobné problémy nastávají i v případech, kdy vývojář používá nástroj, který všechny jeho skripty skládá do jednoho souboru.
- **Dynamický typový systém** – přináší nutné přetypovávání za běhu podle potřeby, které může vést k neočekávanému chování. S tím spojenou nevýhodou je, že datový typ proměnné může být kdykoliv změněn přiřazením hodnoty proměnné, neboť toto chování znesnadňuje funkce nástrojů automatické kontroly, ladění a napovídání (Fenton, 2017, s. xxii-xxv).

Jak již bylo zmíněno JavaScript má mnoho odlišností od ostatních standardních jazyků například z rodiny jazyka C, i přes to, že má podobnou syntaxi. Tyto odlišnosti často vedou ke zmatení vývojářů a k neočekávanému chování jejich programů, a proto opět může být

vhodnější používat alternativy, které například upravují JavaScript více do podoby ostatních jazyků. Základními odlišnostmi jsou:

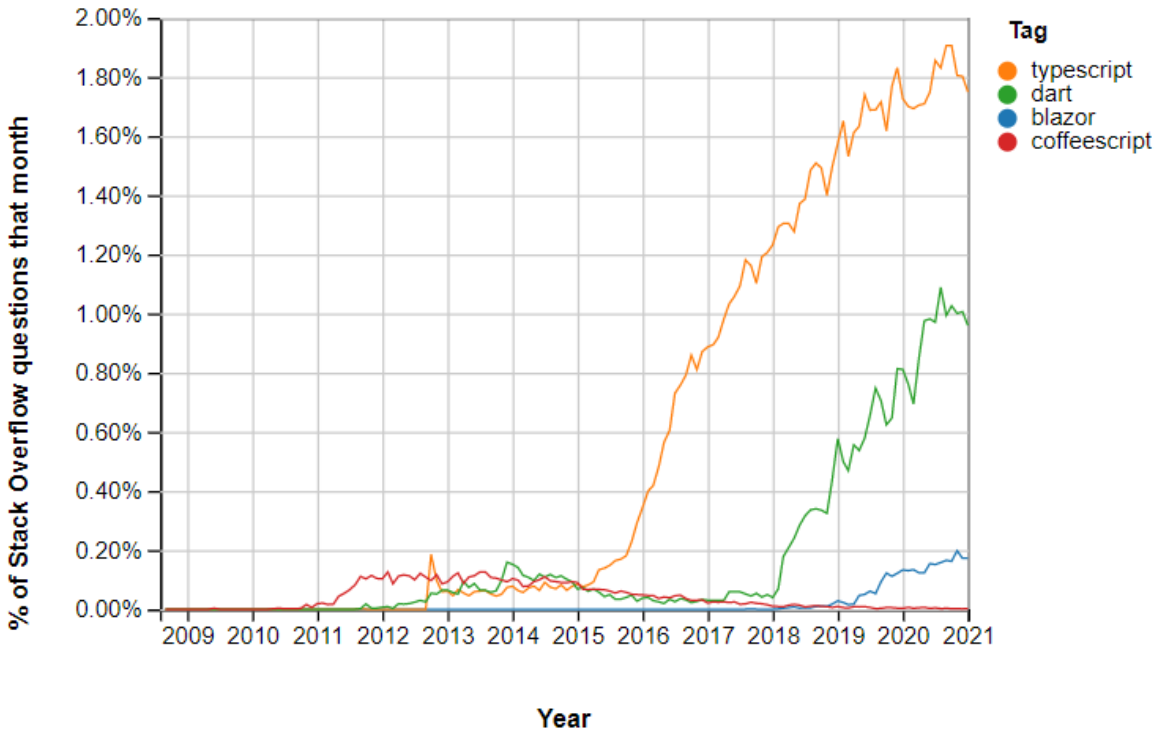
- **Prototypové dědičnosti** – JavaScript je jediným široce používaným jazykem s tímto typem dědičnosti, ostatní standartní jazyky používají dědičnost třídní. I přes to, že prototypová dědičnost přináší spousty výhod, často mate vývojáře. Každý objekt tedy obsahuje objekt prototypu. Pokud chce například příkaz použít vlastnost objektu, kterou nemá, tak se tato vlastnost hledá v prototypech, dokud vlastnost není nalezena, nebo prototyp objektu nemá hodnotu „null“. Toto hledání může být při velkém počtu prototypů náročné na výkon. JavaScript taktéž zachází jinak s konstruktory a s klíčovým slovem „this“.
- **Vše v JavaScriptu je objekt** – jednotlivé funkce, pole a proměnné jsou objekty a podle toho se i chovají. S tím souvisí například to, že nejrychlejším typem cyklu pro procházení pole je obyčejný příkaz „for“.
- **Absence jmenných prostorů** – celý kód se nachází v jednom globálním prostoru, to může vést například k přepisování proměnných. Na rozdíl od ostatních jazyků také složené závorky nevytváří prostor pro lokální proměnné, jediné lokální prostory se vyskytují vně funkcí.
- **Dvě hodnoty označující nic** – jedná se o „null“ a „undefined“. Undefined se chová jako null v ostatních programovacích jazycích, má tedy větší využití a může vždy nahradit null.
- **Automatické vkládání středníků** – JavaScript středníky potřebuje pro ukončování příkazů, tento nástroj může však v některých případech způsobit neočekávané chování programu (Wetzel, nedatováno).

1.5.4 Alternativy JavaScriptu

Alternativ k jazyku JavaScript bylo vytvořeno od jeho vzniku velké množství, a to hlavně z důvodu, jak již bylo zmíněno, jeho nedostatků a odlišností. Pro jejich běh v prohlížeči je nutné, aby obsahovaly transpilátor do JavaScriptu, nebo kompilátor do formátu WASM. Některé alternativy dokonce obsahují transpilátor i kompilátor, a tak si vývojář může vybrat. Mezi nejvýznamnější alternativy patří jazyky TypeScript, Dart, CoffeeScript, Elm a C# s jeho jedním z nejpokročilejších frameworků pro WebAssembly, Blazor.

Jejich přibližnou popularitu mezi vývojáři lze například zjistit pomocí počtu dotazů na vývojářském webu „stackoverflow.com“, viz graf na obrázku 5, vygenerovaném právě na tomto webu. Graf zobrazuje průběh poměru počtu dotazů na daný jazyk vůči všem dotazům na webu během několika let. Dle grafu lze vidět, že jazyk CoffeeScript byl první alternativou mezi těmito nejvýznamnějšími alternativami a při svém vzniku se pyšnil velké oblibě, ta však se vznikem ostatních alternativ začala klesat. Podobný průběh jako CoffeeScript má i jazyk Dart, ten však časem nabyl obrovské popularity díky svému frameworku Flutter. Jelikož je TypeScript nadmnožinou JavaScriptu, která ho pouze obohacuje o další funkcionality, je jednoduché předělat projekt v JavaScript do projektu v tomto jazyce a je tak velmi oblíbený. Nejnovější alternativou je právě framework Blazor. Ten je v současné době stále poměrně novinkou a dle jeho rostoucího průběhu v grafu lze usoudit, že se také jedná o velmi oblíbenou alternativu, která však zatím neměla čas se tolik prosadit jako ostatní alternativy. Komunita jazyka Elm je velmi malá a není tak možné jazyk zobrazit v tomto grafu. Pokud bychom vynesli na graf i poměr dotazů na

jazyk JavaScript, jeho průběh by zcela zastínil všechny alternativy. V posledních letech se vyskytuje průměrně v 11 % všech dotazů na webu a jedná se tak o jeden z nejpopulárnějších jazyků.



Obrázek 5 – Popularita jednotlivých alternativ dle počtu dotazů na webu stackoverflow.com (StackOverflow, 2021)

1.6 CoffeeScript

CoffeeScript je malý jazyk, nutný transpilovat do JavaScriptu. Vznikl přímo za cílem zvýraznit lepší části JavaScriptu a zakrýt ty horší. Výhodou jeho transpilace je možnost používat i kód, napsaný v JavaScriptu. Aktuální verze jazyka je 2.5.1, plně podporující standard ECMAScript 6. Syntaxe je inspirována jazyky Python a Ruby a implementuje i některé jejich funkcionality. Částečně má podobnou syntaxi také se samotným JavaScriptem, nejedná se však o nadmnožinu jazyka JavaScript, tedy kód v JavaScriptu není validním kódem pro CoffeeScript. Navržený jako nadmnožina není kvůli využívání mezer jako části kódu a kvůli vyvarování se zastaralým a zvláštním částem JavaScriptu. Nicméně, tak jako je to u JavaScriptu, není jeho použití určeno pouze na klientskou stranu webových aplikací a lze jej využít i pro skriptování na serveru, například pomocí technologie Node.js. Kód v CoffeeScriptu se zapisuje do samostatného souboru s příponou „.coffee“ (Maccaw, 2012).

1.6.1 Historický vývoj

Počátky jazyka sahají až do roku 2009, kdy Jeremy Ashkenas přišel s novým jazykem, který se transpiluje do JavaScriptu. Hlavními myšlenkami projektu byl jazyk lépe čitelný, jednodušší a méně náchylný na chyby než JavaScript. Představil například jako první zápis funkce znakem šipky, namísto klíčového slova. Tato funkcionalita se časem dostala i do standardu ECMAScript 6. Již v počátcích se stal velmi populárním projektem na platformě GitHub. Měl velké ambice k tomu, aby nahradil JavaScript, situaci výrazně změnilo až vydání standardizace ECMAScript 6, která přinesla spousty inovací a převzala do JavaScriptu mnoho funkcionalit právě z CoffeeScriptu. Tím popularita CoffeeScriptu začala značně klesat a dnes je jedním z nejméně používaných jazyků (Zaczek, 2020).

1.6.2 Výhody a nevýhody

Jeho hlavní výhodou a taktéž důvodem vzniku je zakrytí odlišných částí JavaScriptu od standardních jazyků, čímž snižuje pravděpodobnost vzniku neočekávaných chyb. Přední výhodou je také úspornost zápisu, údajně mohou mít aplikace napsané v CoffeeScriptu až o polovinu kratší zápis než aplikace vytvoření v JavaScriptu. Úspornost jazyka je hlavně zapříčiněna využíváním mezer namísto složených závorek pro vymezení bloku kódu. Jazyk navíc proti JavaScriptu obsahuje předpřipravené často používané funkcionality vedoucí taktéž ke zjednodušení a zkrácení zápisu, jde primárně o jednodušší práci s poli a třídami. CoffeeScript lze díky map ladit přímo v prohlížeči.

Mezi nevýhody jazyka, tak jako u všech transpilovaných alternativ patří přítomnost dalšího kompilačního prvku mezi vytvořením zdrojovým kódem a JavaScriptem. Taktéž jeho nesprávné užívání může vést ke snížení výkonu. Nejvýznamnější nevýhodou jazyka je jeho zmenšující se komunita. Dnes jde o jednu z nejméně používaných alternativ pro nové projekty a vyskytuje se spíše z historických důvodů u starších projektů. Méně zajímavou alternativu z něj také činí skutečnost, že řeší mnohem méně důležitých problémů JavaScriptu než ostatní alternativy (Maccaw, 2012).

1.6.3 Základní vlastnosti a syntaxe

Jelikož vychází z JavaScriptu, jedná se o syntaxí podobný jazyk, využívající prototypovou dědičnost a další typické vlastnosti. Přidává i nové funkcionality, některé části JavaScriptu úplně zakrývá a dělá syntaxi mnohem kratší a jednodušší. Významnými rozdíly pro syntaxi je nahrazení složených závorek mezerami a tabulátory. Komunikace s kódem v JavaScriptu nevyžaduje žádný specifický přístup a je možné používat JavaScriptový kód stejně jako kdyby byl napsaný v CoffeeScriptu (Maccaw, 2012).

1.6.3.1 Komentáře

Jazyk umožňuje zápis jednořádkových komentářů použitím znaku „#“ a víceřádkových komentářů uvnitř bloku začínajícím a končícím třemi znaky „#“ (Maccaw, 2012, s. 1-2).

```
#komentář
###
víceřádkový komentář
###
```

1.6.3.2 Proměnné

Jazyk pro tvorbu proměnných záměrně nepoužívá klíčové slovo var z JavaScriptu, neboť může vést k zastínění proměnných, bere se tedy jako zvláštní část jazyka, kterou CoffeeScript záměrně zakrývá. Proměnnou tedy CoffeeScript obaluje anonymní funkcí tak, aby nebylo možné jí zastínit a aby uchovávala lokální kontext. Jazyk umožňuje pouze lokální proměnné, a tak není možné omylem vytvořit globální proměnnou.

```
prom = "text"
```

Někdy je však výhodné používat globální proměnné, v tomto případě lze globální proměnnou vytvořit jako vlastnost globálního objektu prohlížeče „window“, nebo je možné vytvořit vlastní objekt, který bude stejně globální jako objekt prohlížeče window (Maccaw, 2012, s. 2).

```
window.var = "text"
exports = this
exports.var = "text"
```

1.6.3.3 Funkce

Pro deklaraci funkce se namísto klíčového slova „function“ používá znak šipky „->“. Funkce mohou být jednořádkové i víceřádkové. Poslední výraz uveden v bloku funkce je implicitně návratovou hodnotou funkce, nemusí se tedy používat klíčové slovo „return“.

```
nasobeni = (a,b) -> a * b
funkce = ->
    #řádek
    "text"
```

Volání funkce se může používat stejně jako v JavaScriptu, nebo pokud obsahuje alespoň jeden vstupní parametr, může se používat volání funkce převzaté z jazyka Ruby, tedy bez kulatých závorek a s mezerou mezi názvem funkce a parametry (Maccaw, 2012, s. 2-4).

```
alert()
alert(5)
alert 5
```

1.6.3.4 Objekty a pole

Objekty lze tvořit pomocí stejného zápisu jako v JavaScriptu. Je však možný i zápis bez uvozovek, či dokonce bez čárek mezi atributy objektu.

```
objekt1 = {vlastnost1: "a", vlastnost2: 3}
objekt2 = vlastnost1: "a", vlastnost2: 3
objekt3 =
    vlastnost1: "a"
    vlastnost2: 3
```

Pole má opět stejný zápis jako v JavaScriptu, přičemž umožňuje i variantu zápisu bez čárek mezi hodnotami (Maccaw, 2012, s. 4-5).

```
pole1 = [1,2,3]
pole2 = [
    1
    2
]
```

1.6.3.5 Kontrola toku kódu

Příkazy pro kontrolu toku kódu se jazyk značně liší od JavaScriptu. Podmínku „if“ je možné zapsat víceřádkovým způsobem, či jednořádkovým, přičemž je nutné ještě použít klíčové slovo „then“. Příkaz „if“ je také možné zapsat až za příkaz, který bude proveden, pokud bude podmínka splněna. Booleovské operace se namísto pomocí znaků zapisují slovně, jedná se tedy o funkce „and“, „or“, „not“, „is“, a „isnt“. CoffeeScript navíc obsahuje podmíněný příkaz „unless“, který při nesplnění podmínky umožní provádět příkazy v těle podmíněného příkazu (Maccaw, 2012, s. 5-6).

```
if num1 > num2 then "větší" else "menší"

if num1 > num2
    "větší"
else
    "menší"

alert num1 if num1 < 5

unless num1 > num2
    "měnší"
```

1.6.3.6 Cykly

Jazyk obsahuje dva druhy cyklů. První cyklus se zapisuje pomocí klíčových slov „for“ a „in“, jedná se o moderní cyklus známý z objektově orientovaného programování, umožňující postupné procházení objekty, poli a rozsahy. Tento cyklus umožňuje v těle získat index iterace, filtrace, iterace v atributech objektu a lze jej zapsat v jednom řádku. Při překladu se převede na JavaScriptový cyklus „for“, takže tento cyklus je dokonce rychlejší než obdobný cyklus „forEach“ ze standardu ECMAScript 5. Druhý cyklus je „while“, který navíc obsahuje pole výsledků jako výstupní parametr (Maccaw, 2012, s. 6-7).

```
for jmeno, index in ["Martin", "Jan", "Jindra"]
    alert "#{index} #{jmeno}"
```

1.6.3.7 Třídy

Třídy se definují klíčovým slovem „class“, přičemž mohou obsahovat atributy, a dokonce i konstruktor. Pokud název parametru v konstruktoru začíná znakem „@“, bude stejnojmenný atribut v nové instanci třídy rovnou obsahovat jeho hodnotu zadanou při tvorbě nové instance. Pokud atribut začíná znakem „@“, jedná se o statický atribut, který je možný používat bez tvorby instance třídy.

```
class Osoba
    jmeno: "Tom"
    vypisJmeno: -> alert jmeno
osoba = new Osoba
```

Dědění třídy je možné implementovat použitím klíčového slova „exnteds“. Jelikož však jazyk obsahuje prototypovou dědičnost a všechny třídy jsou dynamické, pokud bude za běhu rodičovské třídě přidán atribut, všechny dědící objekty budou obsahovat také tento atribut (Maccaw, 2012, s. 9-15).

1.7 TypeScript

TypeScript je jazyk společnosti Microsoft, inspirovaný jazykem C# vydaný pod open source licencí Apache 2.0. Jedná se o typovou nadmnožinou jazyka JavaScript, která zároveň upravuje jeho strukturu a obsahuje velké množství nových funkcionalit řešících jeho nevýhody a odlišnosti. Z toho vyplívá validita JavaScriptového kódu pro TypeScript. Mezi hlavní cíle jazyka patří upravení struktury JavaScriptu tak, aby byl jazyk vhodný pro rozsáhlé projekty obsahující tisíce řádků a zlepšení rozvržení a přehlednosti projektů, aby byla práce programátorů rychlejší a efektivnější. Kontroluje také transpilaci projektů a snaží se zlepšit dynamické načítání modulů za běhu. Významným rozdílem proti JavaScriptu je používání statických datových typů. Transpilace do JavaScriptu přináší i své výhody, jako že je například tak TypeScript multiplatformním jazykem, neboť jeho aplikace lze spouštět v prohlížeči, na serveru, a dokonce i jako nativní aplikace pomocí knihoven jako je WinJS. Schopnosti jazyka, Microsoft vyzkoušel tvorbou významných velikých aplikací jako je Azure Management Portal, s projektem obsahujícím přes milion řádků kódu a tvorbou textového editoru Visual Studio Code s projektem obsahujícím přes 300 tisíc řádků. Jazyk umožňuje používat všechny JavaScriptové knihovny a frameworky. Stejně jako CoffeeScript může být provázán mapami a lze jej ladit přímo v prohlížeči. V současné době se pyšní velkým ohlasem, způsobeným jeho značnou podobností s jazyky rodiny C. Soubor obsahující kód v TypeScriptu musí mít příponu „.ts“.

Funkcionality jazyka se děli do tří částí, kde první dvě částí jsou závislé na standardu ECMAScript a obsahují všechny jeho funkcionality, jedná se o většinu funkcionalit jazyka. Do třetí části patří funkcionality, které JavaScript neobsahuje, jedná se například o generické typy a statickou typovou notaci. TypeScript se kompletně skládá ze tří následujících oddělených komplementárních částí:

- **Jazyk** – jedná se konkrétně o syntaxi, klíčová slova, statické datové typy a další.
- **Kompilátor** – vede transpilaci jazyka do JavaScriptu, při které musí v projektu smazat datové typy a vyhledávat chyby. Obsahuje i doplňkové nástroje sloužící pro kompilaci všech skriptů do jednoho výsledného souboru, generování zdrojových map a další.
- **Služby jazyka** – obsahují informace pro vývojářské nástroje, vývojová prostředí a textové editory (Fenton, 2017, s. xix-xxii).

Existuje také upravená verze TypeScript určená pro kompilaci do formátu WASM, zvaná „AssemblyScript“.

1.7.1 Historický vývoj

Vývoj TypeScriptu vedl stejně jako vývoj jazyka C# Anders Hejlsberg a byl poprvé vydán ve verzi 0.8 v roce 2012 jako open source projekt. Rok na to byl jazyk zpřístupněn pro většinu nejpoužívanějších vývojových prostředí a byly do něj doplněny generické typy. V roce 2014 byla vydána verze 1.0, která obsahovala nový rychlejší kompilátor. Postupem času byly do jazyka přidávány další funkcionality, jako jsou například párové struktury „tuple“. Současná verze jazyka je 4.0. (Cleverism, nedatováno).

1.7.2 Výhody a nevýhody

Jelikož se jedná o jazyk transpilovaný do JavaScriptu, platí pro něj stejná omezení jako pro JavaScript a TypeScript nemůže nijak měnit jeho vnitřní chování. Jazyk se zaměřuje zejména na nevýhody ohledně struktury kódu JavaScriptu a upravuje ji tak aby byl jazyk vhodný pro rozsáhlé projekty, v této oblasti tedy konkrétně upravuje například jmenné prostory, vzhled kódu a správu modulů. Kromě nevýhod JavaScript se zaměřuje i na jeho odlišnosti proti standardním jazykům rodiny C, které kvůli podobnosti syntaxe s jazykem Java matou vývojáře. TypeScript se zbavuje těchto odlišností a nahrazuje je standardními řešeními, jako je třídní dědičnost, statické datové typy a další. TypeScript tedy řeší hlavně níže zmíněné problémy:

- **Prototypová dědičnost** – jazyk ji nahrazuje třídní dědičností a přidává tak třídy, jmenné prostory, moduly a rozhrání. To umožňuje vývojářům využít jejich znalosti z ostatních programovacích jazyků a zbavuje je nutnosti znát prototypovou dědičnost.
- **Dynamické typy** – TypeScript obsahuje dobrovolné statické datové typy, díky kterým lze jednoduše kontrolovat operace s jednotlivými proměnnými daných datových typů a jejich konverze. Díky statickým typům mohou být také vývojářské nástroje pro automatickou kontrolu, ladění a napovídání mnohem více nápomocné.
- **Prostory** – TypeScript varuje, při nevhodném užívání prostorů vývojáře, o odlišném chování prostorů v JavaScriptu.
- **Management modulů** – obstarává jazyk pomocí funkcionalit pro načítání modulů (Fenton, 2017, s. xxii-xxv).

Na druhou stranu může být proti JavaScriptu nevýhodné používat TypeScript na menší projekty. Jazyk nepodporuje všechny JavaScriptové frameworky, jako je například EmberJS. Práce s ním může být také složitější, neboť vyžaduje určité nástroje a vývojář musí kód převádět do JavaScriptu (Nance, 2017).

1.7.3 Základní vlastnosti a syntaxe

Jelikož je TypeScript nadmnožinou JavaScriptu, neliší se ve většině vlastností, důkazem je právě validita JavaScriptového kódu pro TypeScript. Obsahuje tedy všechnu syntaxi z jazyka JavaScript, a navíc přidává i svou vlastní. Největší rozdíly jsou v třídní dědičnosti a datových typech. Podporuje také definiční soubory sloužící jako informační soubor o nějaké používané JavaScriptové knihovně, popisující její strukturu, podobně jako hlavičkové soubory v jazyce C++. Pokud tedy chceme využívat bezpečně kód napsaný v JavaScriptu je nutné jej popsat v definičním souboru, můžeme ho využívat však i bez tohoto souboru. Proto nyní budou popsány především vlastnosti syntaxe, které obsahuje TypeScript proti JavaScriptu navíc (Fenton, 2017, s. 1-2).

1.7.3.1 Proměnné

Proměnné je doporučováno deklarovat pomocí klíčového slova „let“, neboť proměnné deklarované tímto způsobem na rozdíl od těch deklarovaných klíčovým slovem „var“ respektují prostory bloků vytvořených složenými závorkami a znemožňují přepsat globální proměnnou nějakou lokální (Fenton, 2017, s. 4-5).

```
let cislo = 2;
{
    let cislo = 3;
    console.log(cislo);
}
console.log(cislo);
```

1.7.3.2 Statické datové typy

Jazyk obsahuje základní statické typy jako je „string“ pro textové řetězce, „number“ pro čísla, „boolean“ pro logické hodnoty a „symbol“ či „any“ pro dynamické typy. Speciálními typy jsou „undefined“ pro hodnotu proměnné bez přiřazené hodnoty, „null“ pro značení absence hodnoty objektu, „void“ pro případy, kde není žádná hodnota a „never“ pro značení nedosažitelné části kódu.

Datové typy se zapisují na rozdíl od většiny programovacích jazyků až za jméno proměnné, čímž jazyk poukazuje na dobrovolnost jejich zápisu.

Pokud nebude proměnné zadán žádný datový typ, bude mu automaticky vnitřně vygenerován podle typu přiřazené hodnoty, aby nástroje jazyka mohli vývojáři poskytovat nápovědu.

```
//jednoduché typy
const jmeno: string = "Jan";
const vyska: number = 186;
const aktivni: boolean = true;

//pole
const jmena: string[] = ["Josef", "Jan", "Tereza"];

//funkce s parametrem a návratovou hodnotou
let pozdrav = (jmeno: string): string => {
    return "Ahoj " + jmeno;
};

//objekt
let osoba: {
    jmeno: string;
    vyska: number;
};
```

Pro tvorbu vlastních složitých datových typů lze použít rozhrání, či typový alias (Fenton, 2017, s. 6-9).

```
//rozhrání
interface IOsoba {
    jmeno: string;
    vyska: number;
}
const Tom1: IOsoba = {
    jmeno: "Tom",
    vyska: 183
}
//typový alias
type OsobaTyp = {
    jmeno: string;
    vyska: number;
};
const Tom2: OsobaTyp = {
    jmeno: "Tom",
    vyska: 169
};
```

1.7.3.3 Enumerátory

Jazyk dále obohacuje JavaScript i o typ pro enumerátory, který JavaScript nemá definovaný a musí se v něm vytvářet jiným způsobem (Fenton, 2017, s. 10).

```
enum TypZarizeni {
    Desktop,
    Tablet,
    Telefon,
}
const zarizeni = TypZarizeni.Desktop;
const nazevZarizeni = TypZarizeni[zarizeni];
```

1.7.3.4 Třídy

Třídy jsou zde velmi podobné jako v jazyce C#. Jazyk umožňuje vytvářet třídy s atributy a konstruktorem. Atributy obsahují přístupové modifikátory, určující, kdo má k daným atributům přístup. Mezi přístupové modifikátory patří modifikátor „private“, pro atributy viditelné pouze vně třídy, modifikátor „protected“, pro atributy viditelné vně třídy a v dědících třídách a modifikátor „public“, pro objekty viditelné vždy. Také lze vytvářet abstraktní třídy, ze kterých nelze vytvářet instance a jsou určené, aby z nich jiné třídy dědily.

```
class Osoba {

    private jmeno: string;
    private vyska: number;

    constructor(jmeno: string, vyska: number) {
        this.jmeno = jmeno;
        this.vyska = vyska;
    }
    public VypisInformace(): void {
        alert(`Jmenuji se ${this.jmeno} a měřím ${this.vyska}cm`);
    }
}
```

Třída může dědit z jiné třídy pomocí klíčového slova „extends“, nebo z rozhrání pomocí klíčového slova „implements“ (Fenton, 2017, s. 47-53).

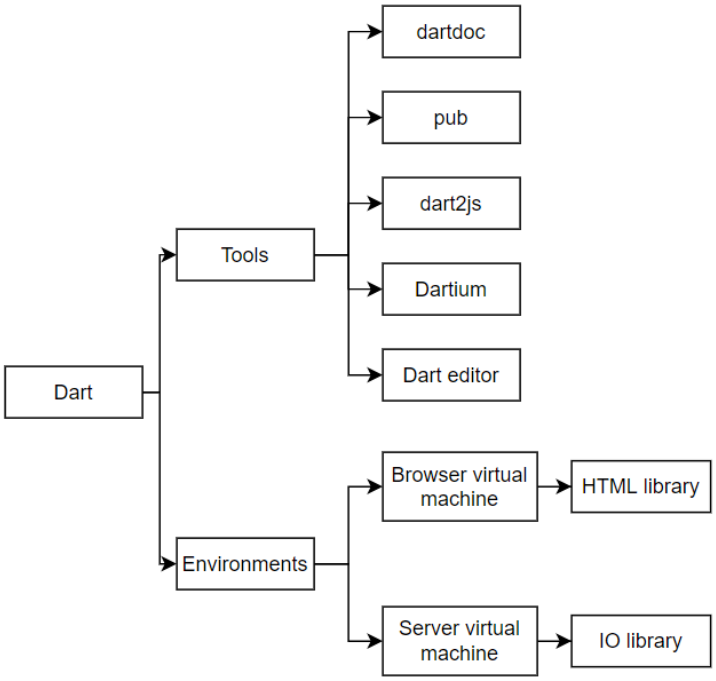
```
interface Informace {
    VypisInformace(): void;
}

class Osoba implements Informace {

    private jmeno: string;
    private vyska: number;

    constructor(jmeno: string, vyska: number) {
        this.jmeno = jmeno;
        this.vyska = vyska;
    }
    public VypisInformace(): void {
        alert(`Jmenuji se ${this.jmeno} a měřím ${this.vyska}cm`);
    }
}

class Administrator extends Osoba {
    private opravneniZapisovat:boolean;
    constructor(jmeno: string, vyska: number, opravneniZapisovat:boolean) {
        super(jmeno,vyska);
        this.opravneniZapisovat = opravneniZapisovat;
    }
}
```

1.8 Dart

Dart je open source objektově orientovaný programovací jazyk společnosti Google, který je optimalizovaný pro tvorbu komplexních aplikací. Cílem jazyka je být nástrojem pro jednoduchý vývoj aplikací s vysokým výkonem. Dart je multiplatformní, lze jej tedy využít přímo v prohlížečích s podporou Dartu (Dartium), v ostatních prohlížečích pomocí transpilace do JavaScriptu, na serverech, či pro tvorbu nativních aplikací.

Dart je v mnoha ohledech velmi podobnou alternativou JavaScriptu jako TypeScript. Je také inspirovaný jazyky rodiny jazky C a využívá jejich funkcionality, tedy zejména třídní dědičnost a statické datové typy. Stejně jako TypeScript přináší podobné výhody jako strukturu vhodnou pro rozsáhlé projekty, správu modulů a další. Na rozdíl od TypeScriptu se však jedná o zcela odlišný jazyk od JavaScriptu a není jeho nadmnožinou. Soubor jazyka Dart má příponu „.dart“.

Dart mimo jazyku obsahuje kompletní platformu pro vývoj aplikací. Platforma obsahuje konkrétně nástroje pro vývoj aplikací, nástroje pro překlad do JavaScriptu, editor a běhová prostředí pro aplikace vytvořené v Dartu (Buckett, 2013, s. 3-5). Schéma ekosystému Dartu je na obrázku 6.



Obrázek 6 – Ekosystém Dartu (Buckett, 2013, s. 4)

V současné době se Dart pyšní velké popularitě díky novému frameworku „Flutter“, umožňující vývoj výkonných nativních aplikací (Bolton, 2019).

1.8.1 Historický vývoj

První verze Dartu byla vydána v roce 2013 s cílem nahradit JavaScript v prohlížečích. Tato verze obsahovala vlastní virtuální stroj umožňující běh nativních aplikací v prohlížeči Chrome. Jazyk vznikl v Googlu pravděpodobně kvůli problémům s JavaScriptem při tvorbě programů Gmail a Google mapy. Google také vydal speciální prohlížeč Dartium

s podporou Dartu. I přes velký marketingový tlak na vývojáře, aby používali Dart, zůstali u JavaScriptu a Dartu se jej tedy nikdy nepovedlo nahradit.

Kvůli neúspěchu s nahrazením JavaScriptu ztrácel Dart na slávě a před úplným úpadkem jej zachránil až nový UI framework od Googlu zvaný Flutter. Flutter umožnil Dartu zejména tvorbu nativních aplikací pro operační systémy Android a iOS. Google krátce poté začal vyvíjet v jazyce Dart nový operační systém „Fuchsia“, který by v budoucnu možná mohl nahradit systémy Android a ChromeOS (Bolton, 2019).

1.8.2 Výhody a nevýhody

Dart má podobné výhody a nevýhody jako má TypeScript. Jelikož Dart pro webové aplikace není tak populární jako TypeScript je jeho nevýhodou také menší komunita a z toho vyplívající i menší množství informačních a studijních materiálů a menší ekosystém komunitních knihoven, balíčků a frameworků. Nevýhodou může být také nutnost učit se nový programovací jazyk, který je však na druhou stranu velmi podobný jazykům rodiny jazyka C (Rezvi, 2020).

1.8.3 Základní syntaxe

Syntaxe jazyka je velmi podobná jazykům C#, JavaScript a Java, jazyk tedy využívá třídní dědičnost a statické datové typy. Níže jsou popsány zejména odlišnosti v syntaxi od jazyků C# a Java a nejvýznamnější funkcionality jazyka.

1.8.3.1 Proměnné

Dart využívá statické datové typy, ale umožňuje i dynamické. Pro základní statické typy obsahuje klíčová slova „int“, „String“, „double“, „bool“ a „num“ a pro dynamické „var“ a „dynamic“. Pro tvorbu polí, lze využít třídu pro seznamy „List“. Seznam řetězců by měl například typ „List<String>“. Pro definování konstanty slouží klíčové slovo „final“ (Buckett, 2013, s. 9-10).

```
String promenna1 = "A";
var promenna2 = 2;
List<String> retezce = new List();
```

1.8.3.2 Funkce Main

Jazyk používá pro začátek vykonávání kódu, tak jako jazyky, kterými se inspiruje, funkci „Main“. To může být pro vývojáře výhodou, protože je definované kde přesně kód začíná (Buckett, 2013, s. 4).

```
void main() {
  print("začátek");
}
```

1.8.3.3 Knihovny

Jazyk obsahuje velké množství standardních knihoven, jedná se například o knihovnu „dart:html“, pro práci s HTML elementy a modelem DOM, „dart:core“, pro základní funkcionality, „dart:math“, obsahující matematické funkce a konstanty a „dart:io“ pro práci se soubory.

Vlastní kód lze rozdělit do souborů a knihoven. Pro přístup ke knihovně, či k souboru je nutné jejich obsah naimportovat pomocí klíčového slova „import“ (Buckett, 2013, s. 13-16).

```
import 'Knihovna.dart';
import 'dart:js' as js;
import 'dart:html';
```

1.8.3.4 Třídy

Třídy fungují opět obdobně jako v jazyce C#, či Java, obsahují tedy atributy a konstruktor a používají třídní dědění. Dart obsahuje jenom modifikátory přístupu public a private. Pro modifikátory však neexistují klíčová slova, ale definují se zápisem názvu atributu. Veřejný atribut se značí prvním velkým písmenem v názvu a privátní atribut podtržítkem na začátku názvu. Pro dědění z třídy slouží klíčové slovo „extends“ a pro dědění z rozhrání klíčové slovo „implements“ (Buckett, 2013, s. 10-13).

```
class Osoba {
  String Jmeno;
  int _vyska;

  Osoba(String jmeno, int vyska) {
    this.Jmeno = jmeno;
    this._vyska = vyska;
  }
  void VypisInformace() {
    print("Jmenuji se ${this.Jmeno} a měřím ${this._vyska}cm");
  }
}


class Administrator extends Osoba {
  bool _opravneniZapisovat;

  Administrator(String jmeno, int vyska, bool opravneniZapisovat)
      : super(jmeno, vyska) {
    this.Jmeno = jmeno;
    this._vyska = vyska;
    this._opravneniZapisovat = opravneniZapisovat;
  }
}
```

1.8.3.5 Práce s HTML

Jak již bylo zmíněno Dart obsahuje standardní knihovnu „dart:html“, kterou je nutno naimportovat do souboru, ve kterém se pracuje s HTML elementy a DOM modelem. Tato knihovna nabízí podobně jednoduchý přístup k HTML elementům, jako nabízí například knihovna jQuery pro JavaScript. V kódu níže je ukázán přístup a modifikace již existujícího elementu a tvorbu zcela nového elementu, přičemž je využít zkrácený zápis pro nastavení vlastností objektu (Buckett, 2013, s. 18-20).

```
querySelector('#Tlacitko1').onClick.listen((e) {
      Funkce1();
});
var tlacitko2 = ButtonElement()
      ..id = 'Tlacitko2'
      ..onClick.listen((e) {
        Funkce2();
      })
      ..classes.add('Button')
      ..title = 'Popisek';
parent.children.add(tlacitko2);
```

1.8.3.6 Komunikace s JavaScriptem

Komunikace s JavaScriptem v Dartu je realizována pomocí přerušení, definovaných ve standardní knihovně „dart:js“. Knihovna obsahuje objekt „context“ odkazující na objekt prohlížeče „window“ skrz, který je možné přistupovat k jednotlivým funkcím a vlastnostem (Buckett, 2013, s. 259-268).

```
import 'dart:js' as js;
js.context["Objekt"].callMethod('Funkce1');
```

1.9 Blazor

Blazor je open source webový „SPA“ (Single Page Application) framework společnosti Microsoft, umožňující vývoj webových aplikací se zpracováním dat na klientské straně pomocí jazyku C#. Na rozdíl od ostatních zmíněných alternativ se nejedná o programovací jazyk, ale o framework pro jazyk C# a místo transpilace do JavaScriptu využívá kompilaci do standardního formátu WASM. Jedná se tedy o podobný framework jako je Angular, React, či Vue pro JavaScript.

Framework získal svůj název spojením anglického slova Browser, tedy prohlížeč a názvu „Razor“, což je HTML „template engine“ .NET frameworku. Název tedy naznačuje, že lze Razor kromě vytváření stránek na serveru využít i pro vytváření stránek přímo v prohlížeči. Jelikož WebAssembly nemá rozhrání pro práci s modelem DOM, obsahuje Blazor pro zpracování modelu virtuální stromovou strukturu, „render tree“, kterou posílá při potřebě aktualizace stránky JavaScriptu, který model DOM aktualizuje.

Blazor nabízí dva modely tvorby aplikací. Práce s oběma modely je téměř totožná, první model byl již naznačený „WebAssembly Blazor“ využívající formát WASM a zpracovávající data plně na klientské straně. Druhý model vůbec nepoužívá technologii WebAssembly, jedná o model „Server-Side Blazor“. Tento model zpracovává všechna data na serveru a komunikuje s prohlížečem pomocí technologie „SignalR“. Při používání tohoto modelu je kladena značně vyšší náročnost na server a také aplikace musí mít vždy přístup k serveru, nefunguje tedy v režimu offline (Himshoot, 2019, s. xv-xxiii).

Kromě Blazoru může být pro podporu WebAssembly v jazyce C# použit i například komunitní framework „Lara“, nebo může být jazyk C# dokonce přeložen do JavaScriptu pomocí nástroje Bridge.NET.

1.9.1 Historický vývoj

Blazor byl vytvořen jako osobní experimentální projekt vývojářem Microsoftu Stevem Sandersonem a prvně byl představen roku 2017. Dříve využíval knihovny platformy „Mono“ od Microsoftu. Platforma Mono umožňuje jazykům frameworku .NET vytvářet nativní aplikace pro operační systémy Windows, Android, iOS a Linux pomocí technologie Xamarin a hry pomocí technologie Unity. V současné době je Blazor plnohodnotnou součástí frameworku .NET a využívá pro svůj běh právě knihovny tohoto frameworku, díky čemuž je nyní značně výkonnější (Himshoot, 2019, s. xv-xix). V současné době se také jedná o jeden z nejvíce pokročilých WebAssembly nástrojů (Drozdík, 2019).

1.9.2 C# a .NET framework

Jazyk C# je jednoduchý výkonný objektově orientovaný multiplatformní programovací jazyk, jehož vývoj je veden vývojářem Andersem Hejlsbergem ve společnosti Microsoft. Z názvu lze vydedukovat, že se jedná o jazyk založený na jazycích rodiny jazyka C. Jazyk obsahuje velké množství funkcionalit a různá programovací paradigmata. V jazyce je možné vytvářet nativní aplikace pro velké množství operačních systémů, hry, webové aplikace a další.

Je také společně s vývojovým prostředím Visual Studio, virtuálním strojem (CLR) a knihovnami součástí .NET frameworku. Kompilátor jazyka C# nejprve převádí kód do

mezikódu „CIL“ (Common Intermediate Language), který využívají všechny jazyky frameworku .NET, tedy i F# a Visual Basic. Virtuální stroj pak slouží k interpretaci mezikódu do instrukcí fyzického procesoru. Standardní knihovny ve frameworku jsou moderní, kvalitní a velmi obsáhlé, obsahují struktury a komponenty potřebné pro práci s různými technologiemi. Framework obsahuje čtyři rozdílné verze, konkrétně Mono, .NET 5, .NET Framework a .NET Standard. Mono vyvíjí společnost Xamarin se stejnojmennou technologií, umožňující vývoj aplikací pro Android a iOS, Mono je však možné využít i pro tvorbu her pomocí nástroje Unity a pro tvorbu Linuxových aplikací. Framework .NET 5 (dříve označován jako .NET Core) je určen pro vývoj multiplatformních a webových aplikací. Původní .NET Framework je určený pro vývoj aplikací pro operační systém Windows. Poslední verzí je .NET Standard, obsahující všechny funkcionality, které jsou pro všechny ostatní frameworky totožné a jsou platformě nezávislé. Zajišťuje tak přenositelnost kódu mezi jednotlivými verzemi frameworku (Čápka, nedatováno).

1.9.3 Výhody a nevýhody

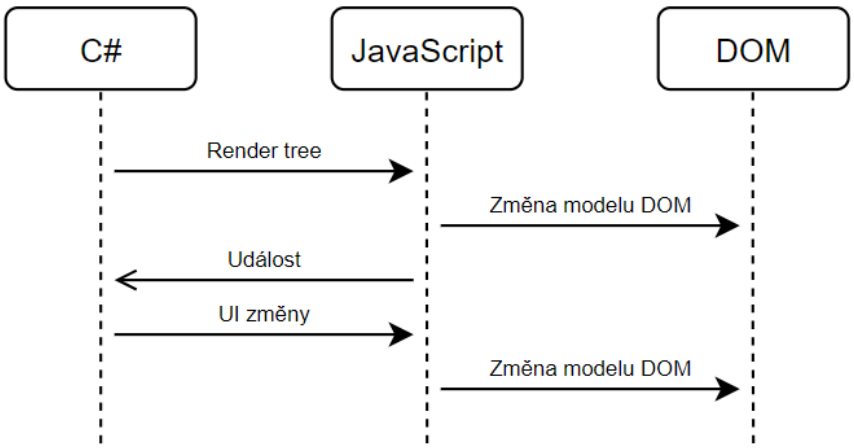
Vymezit výhody a nevýhody Blazoru proti JavaScriptu je mnohem složitější než u ostatních alternativ. Porovnávání znesnadňují skutečnosti, že se jedná o technologii využívající standard WebAssembly, tudíž přebírá všechny jeho výhody a omezení, ale také se jedná o kompletní framework, a tak je vhodnější ho porovnávat spíše s jednotlivými frameworky pro JavaScript.

Možnost využít jazyk C# pro klientské zpracování dat přináší mnoho výhod, neboť se jedná o velmi pokročilý jazyk. Jazyk má dobrou strukturu určenou pro libovolně rozsáhlé projekty a libovolné platformy. .NET framework obsahuje mnoho standardních knihoven, jazyk se však také pyšní velkou komunitou a existuje pro něj i spousty komunitních rozšířeních dostupných zejména v balíčkovacím systému NuGet. Jedná se také o velmi výkonný jazyk. Pro tvorbu aplikací v jazyce C# je dobře optimalizované vývojové prostředí Visual Studio.

Hlavní výhodou, kterou přináší formát WASM je výkon podobný nativním aplikacím, je tedy značně teoreticky výkonnější než JavaScript. Na druhou stranu největší nevýhodou WebAssembly je absence základních rozhrání API a nutnost, tak stejně používat JavaScript například pro aktualizace modelu DOM, čímž WebAssembly naopak ztrácí na výkonu proti JavaScriptu.

Framework Blazor nepřináší žádné nové funkcionality proti ostatním JavaScriptovým frameworkům. Umožňuje však vývojářům používat jazyka C# i pro zpracování dat na klientské straně, vytvářet SPA aplikace, využívat knihovny .NET frameworku a umožňuje také využívat na klientské straně i serverové straně stejný jazyk. Výhodné je tedy použití Blazoru zejména pro vývojáře v jazyce C#, kteří se jeho užitím vyhnout JavaScriptu a jeho frameworkům.

Hlavními nevýhodami je nutnost často využívat jazyk JavaScript, dlouhá načítací doba a velká datová náročnost z důvodu velikost celého frameworku.



Obrázek 7 – Proces generování modelu DOM v Blazoru (Himshoot, 2019, s. xxi)

1.9.4 Základní syntaxe

Syntaxe jazyka C# je typickou syntaxí rodiny jazyka C, proto je níže popsána pouze základní syntaxe spojená s používáním frameworku Blazor.

1.9.4.1 Práce s HTML

Blazor využívá technologii Razor, která umožňuje zapisovat kód v jazyce C# přímo do HTML kódu, přičemž na základě tohoto zápisu se vygeneruje skutečný HTML kód pro prohlížeč. Je tedy možné například mezi HTML elementy vložit podmínku „if“ a do jejího těla vložit HTML elementy, které budou vygenerovány, pokud bude podmínka splněna. Taktéž je možné reagovat na události, či vypisovat data. Tento obohacený HTML kód se zapisuje do souboru s příponou „.razor“. V tomto souboru je možné také vytvořit blok „@code“, který vymezuje prostor pro zápis kódu pouze v jazyce C#. Je možné také místo bloku „@code“ zapisovat kód v jazyce C# do samotného souboru stránky s příponou „.razor.cs“.

```
<button class="Button" @onclick="InkrementaceCisla">
    Tlačítko
</button>
@if (num > 0)
{
    <input type="number" @bind="num" />
}

@code {
    public int num { get; set; } = 0;
    private void InkrementaceCisla() {
        num += 1;
    }
}
```

1.9.4.2 Komponenty

Blazor umožňuje vytvářet vlastní komponenty vystupující jako HTML elementy. Každá stránka s příponou „.razor“ je v Blazoru komponentou a lze ji vložit do jiné stránky. Komponentám je možné také předávat jako u HTML elementů hodnoty atributů, přičemž komponenta musí mít definovaný daný atribut a označený pomocí zápisu „[parameter]“.

```
<div>@text</div>
@code {
    [parameter] public string text { get; set; }
}
```

Kód níže je ukázkou, jak jé možné používat Blazor komponenty stejně jako obyčejné HTML elementy.

```
<Komponenta text="Test"/>
```

1.9.4.3 Životní cyklus

Každá komponenta má svůj životní cyklus, přičemž na každou fázi životního cyklu může vývojář reagovat. Vývojář pro reakci na fázi životního cyklu může využít předdefinované metody, které jsou vyvolány při vstupu do daných fází životního cyklu. Jedná se o metody „OnInitialization“, odkazující na fázi vzniku komponenty, „OnParameterSet“, spuštěná po nastavení hodnot atributů do parametrů, „OnAfterRender“, vyvolaná po překreslení komponenty a „Dispose“, vyvolaná při odstranění komponenty, pokud komponenta implementuje rozhrání IDisposable.

1.9.4.4 Komunikace s JavaScriptem

Kód v jazyce JavaScript je možné používat pomocí přerušení, přičemž je možné k němu přistupovat synchronně i asynchronně. Pro použití je nutné injektovat do komponenty objekt typu IJSRuntime a pomocí tohoto objektu používat jednotlivé atributy nějakého JavaScriptového objektu.

1.10 Další alternativy

Kromě popsaných hlavních čtyř alternativ JavaScriptu existují další méně používané, ale přesto významné alternativy.

1.10.1 Elm

Elm je reaktivní funkcionální programovací jazyk transpilující se do JavaScriptu optimalizovaný pro zpracování dat na klientské straně. Elm klade velký důraz na výkon robustnost a použitelnost. Kromě jazyka obsahuje taktéž celou platformu s nástroji, frameworkem a balíčkovacím systémem. Jazyk se pyšní kvalitní strukturou, funkcionalitami, velmi nápomocným kompilátorem a dalšími nástroji, díky čemuž je při jeho užívání velmi nízká pravděpodobnost vzniku chyb za běhu. Mimo to je Elm mnohem méně datově náročný než ostatní JavaScriptové frameworky a více výkonný. Na druhou stranu může představovat pro vývojáře problém jeho funkcionální paradigmata a malá komunita (Elm, nedatováno).

1.10.2 Babel

I přes to, že se Babel označuje za alternativu, nejedná se o jiný jazyk než JavaScript. Přidává však do kódu funkcionality a takzvané „polyfilly“ umožňující podporu nejnovějších funkcionalit standardu ECMAScript v prohlížečích neobsahujících tyto funkcionality z důvodu implementace starší verze standardu ECMAScript. Jedná se tedy spíše o preprocesor. (Fenton, 2017, s. xxvi).

1.10.3 ClojureScript

ClojureScript je verze funkcionálního jazyka Clojure s transpilátorem do JavaScriptu. Jedná se o jednoduchý a výkonný jazyk, vhodný pro běh v prohlížeči i na serveru. Bohužel jazyk se netěší příliš velké popularitě (Kharchenko, 2018).

1.10.4 Opal

Jedná se o implementaci jazyka Ruby s transpilátorem do JavaScriptu (Kharchenko, 2018).

1.10.5 Kaffeine

Kaffeine je nadmnožina jazyka JavaScript obohacující jej o některé funkce pro efektivnější vývoj (Kharchenko, 2018).

1.10.6 Roy

Roy byl vyvinut jako experimentální nástroj podobný JavaScriptu, avšak s jednodušším zápisem. Také obsahuje některé funkcionality z funkcionálních programovacích jazyků (Kharchenko, 2018).

Vlastní práce

1.11 Návrh aplikace

Při tvorbě demonstrační aplikace je nejprve potřeba vymezit účel a základní vlastnosti aplikace. Po zhotovení prvního kroku je již možné navrhnout i uživatelské rozhrání a strukturu aplikace. Vlastnosti společně s designem budou u všech demonstračních aplikací stejné.

1.11.1 Hlavní myšlenka a vlastnosti aplikace

Hlavní myšlenka demonstrační aplikace zastává roli seznamu úkolů neboli „úkolníčku“. Aplikace tedy slouží k manuální evidenci úkolů uživatele a postupu při jejich plnění. Na trhu se vyskytuje několik takových aplikací, jedná se například o aplikace „Microsoft To Do“, a „Wunderlist“. Výhodou těchto aplikací je, že uživatel si své úkoly nemusí pamatovat a má jasný přehled o svých nesplněných i splněných úkolech. Takové aplikace lze s výhodou využívat například pro vedení časového rozvrhu dne. Pro účely demonstrace zpracování dat na klientské straně různými programovacími jazyky bude stačit jednoduchá statická stránka s prvky webové aplikace. Data budou tedy zpracovávána pouze na klientské straně, zde budou však rovnou i ukládána.

1.11.1.1Hlavní vlastnosti

Dle předchozího popisu hlavní myšlenky, demonstrační aplikace obsahuje následující potřebné vlastnosti. Celá aplikace obsahuje pouze jednu stránku, na které se všechny operace odehrávají.

1.11.1.1.1 Evidence seznamů úkolů

Aplikace v levém postranním panelu obsahuje jeden z hlavních prvků, výpis seznamů úkolů s počty nedokončených úkolů v daném seznamu. Seznam úkolů zastává roli kategorie, do které jednotlivé úkoly patří. V aplikaci jsou dva druhy těchto kategorií, první kategorií jsou seznamy úkolů, které jsou automaticky vygenerované a slučují úkoly ze všech kategorií, které mají nějakou určitou vlastnost. Automaticky vygenerované seznamy obsahuje aplikace čtyři, jedná se o seznam obsahující úkoly s dnešním dnem splnění, seznam s důležitými úkoly, seznam s plánovanými úkoly a seznam se všemi nezařazenými úkoly. Druhý druh seznamů úkolů vytváří sám uživatel a může jim nastavovat jméno, ikonu, či způsob podle kterého se budou řadit úkoly v seznamu. Uživatel tedy může například vytvořit seznam úkolů s názvem „Škola“.

1.11.1.1.2 Evidence úkolů

Po vybrání daného seznamu úkolů se v hlavním prostoru, tedy ve středu aplikace zobrazí všechny úkoly, které seznam obsahuje. Uživatel má možnost úkoly vytvářet, mazat, zadávat jméno, termín splnění, poznámku, kroky postupu a označovat úkoly jako důležité či splněné. Uživatel taktéž může nastavit, pokud chce splněné, či nesplněné úkoly skrýt. Podrobné nastavení úkolu obsahuje pravý postranní panel, který se zobrazí při vybrání některého úkolu.

1.11.1.1.3 Evidence kroků postupu

Pravý postranní panel, mimo nastavení úkolu obsahuje taktéž výpis kroků postupu úkolu. Tyto kroky zastupují jednotlivé postupné činnosti při plnění úkolu. Kroky obsahují popis a lze je označit za splněné.

1.11.1.1.4 Vyhledávání úkolů

Pomocí tlačítka v levém postranním panelu lze zobrazit vyhledávání úkolů. Zde může uživatel vyhledávat úkoly ze všech kategorií, podle zadaného textu do vyhledávacího vstupu.

1.11.1.1.5 Databáze a uživatelské nastavení

V aplikaci je potřeba pro udržení dat ukládat všechna data o seznamech úkolů, úkolech a kroků postupu, které uživatel vytvořil do databáze. Uživatel může mít také nějaké své specifické nastavení aplikace, které je potřeba ukládat do uložiště, zde je pro demonstraci možné ukládat barevný mód aplikace. Data z databáze lze také exportovat do formátu JSON a stáhnout, a zpětně vložit a naimportovat do databáze.

1.11.2 Uživatelské rozhraní

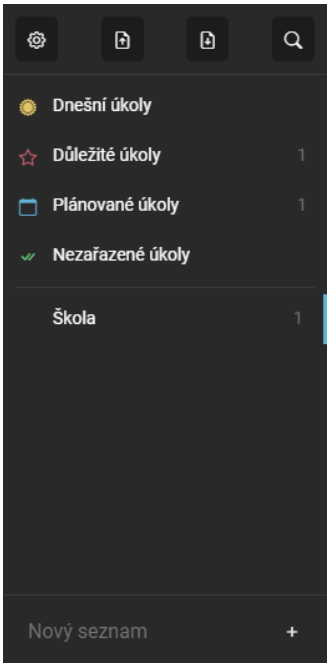
Jelikož se jedná o webovou aplikaci je potřeba pro tvorbu uživatelského rozhraní využít jazyky HTML a CSS, či jiné jejich alternativy. Uživatelské rozhraní bylo vyvinuto pomocí HTML a CSS preprocesoru SCSS. Jak již bylo zmíněno, celá aplikace se odehrává na jediné stránce a je rozdělená do tří následujících částí.

Obrázek 8 – Uživatelské rozhraní demonstrační aplikace (Vlastní zpracování)

1.11.2.1Levý postranní panel

První částí demonstrační aplikace je levý postranní panel. Tento panel obsahuje nahoře tlačítka pro nastavení, manipulaci s daty a zapnutí vyhledávání. Zbytek panelu je věnován seznamům úkolů, které fungují jako kategorie pro dělení jednotlivých úkolů. Každý seznam úkolů má zde vykreslenou ikonu, název a počet nesplněných úkolů v seznamu. Pokud uživatel vybere některý seznam, překreslí se i na hlavní obsah společně s úkoly, které obsahuje. První čtyři seznamy jsou předvytvořené a obsahují úkoly z ostatních

seznamů úkolů v případě, že některá vlastnost daného úkolu splňuje podmínku pro zařazení do tohoto seznamu. Mimo těchto seznamů může uživatel i vytvářet vlastní seznamy pomocí editoru názvu a tlačítka v dolní části panelu.



Obrázek 9 – Levý postranní panel demonstrační aplikace (Vlastní zpracování)

1.11.2.2Seznam úkolů

Hlavní obsah aplikace zobrazuje vybraný seznam úkolů a všechny úkoly, které seznam obsahuje. Zde může uživatel v nastavení přejmenovat seznam, změnit ikonu seznamu a změnit řazení úkolů v seznamu.

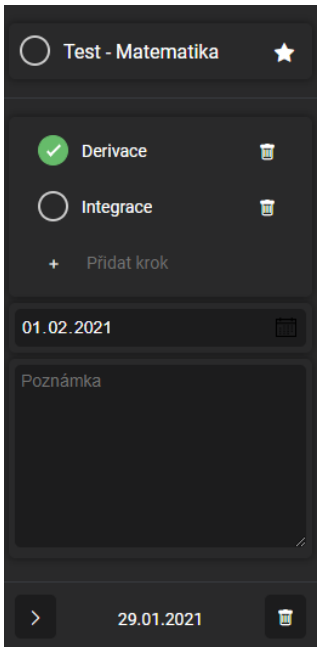
Pod nastavením jsou vykresleny bloky pro splněné a nesplněné úkoly, které je možné pomocí oddělovacích tlačítek skrývat. Zde je možné vidět a nastavovat jeho zadání a zda je úkol splněný, či důležitý. Při kliknutí na nějaký úkol se otevře pravý postranní panel s podrobnějšími nastaveními úkolu.

V dolní části se nachází editor pro zadávání zadání nového úkolu a tlačítko pro přidání nového úkolu.

Obrázek 10 – Hlavní obsah demonstrační aplikace (Vlastní zpracování)

1.11.2.3Pravý postranní panel

Pravý postranní panel obsahuje všechny informace o úkolu. Je zde zobrazeno jeho zadání, zda je splněný, či důležitý, datum splnění, poznámka, datum vytvoření, možnost smazání úkolu a jeho kroky. Kroky úkolu vyjadřují jednoduché menší úkoly, které je potřebné splnit, aby mohl být splněný celý úkol. Těmto krokům lze nastavovat zadání, zda je splněný a je možné je přidávat a mazat.



Obrázek 12 - Pravý postranní panel demonstrační aplikace (Vlastní zpracování)

1.11.2.4Vyhledávání

Další důležitou funkcionalitou je režim hledání úkolů podle toho, zda jejich zadání obsahuje zadaný text. Uživateli je tedy při zapnutí režimu hledání tlačítkem v levém postranním panelu zobrazen editor v horní části hlavního obsahu aplikace pro zadávání hledaného zadání úkolu. Pokud uživatel zadá hledaný text jsou mu okamžitě zobrazeny všechny úkoly jejichž zadání obsahuje daný text.

Obrázek 11 – Vzhled vyhledávání v demonstrační aplikaci (Vlastní zpracování)

1.11.3 Struktura kódu

Když jsou definovány všechny potřebné vlastnosti je možné vytvořit strukturu kódu. Jelikož je daná aplikace vyvíjena vícekrát pomocí odlišných programovacích jazyků pro zpracování dat na klientské straně, není výhodné pro návrh struktury kódu použít třídní model grafického jazyka UML, neboť je závislý na použité platformě a pro každou alternativu by tak měl vypadat jinak. Pro návrh struktury kódu je tak vhodné použít doménový model jazyka UML, který je zobrazený na obrázku 13.

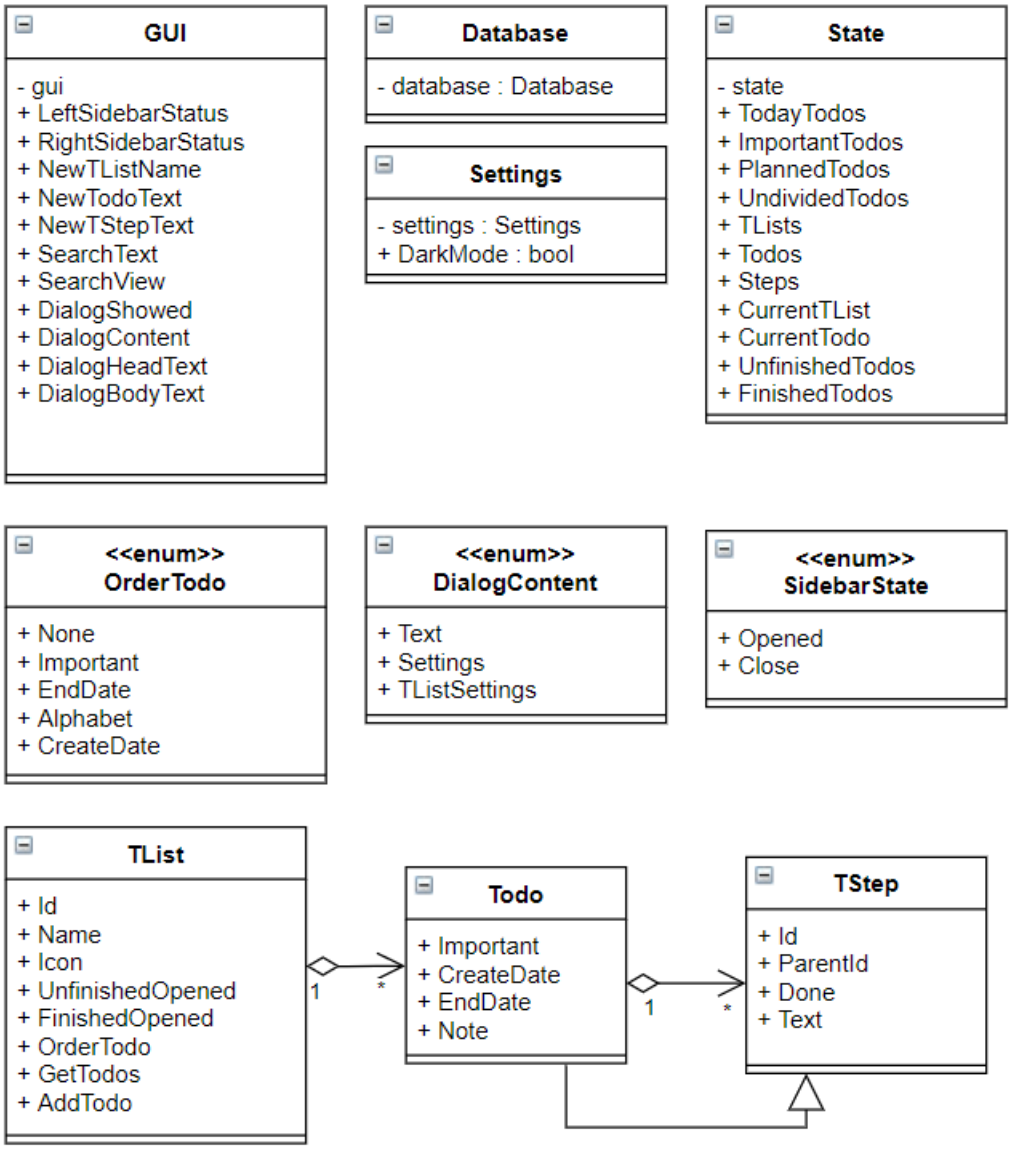
1.11.3.1Uživatelská data

Pro zpracování uživatelských dat jsou určeny třídy „TList“, „Todo“ a „TStep“. Třída TList vyjadřuje jednotlivé seznamy úkolů. Každý seznam může obsahovat neomezené množství úkolů, které vyjadřuje třída Todo. Naopak každý úkol by měl patřit pouze do jednoho seznamu úkolů, mohou být však i přes to úkoly zobrazeny zároveň jako obsah i předvytvořených seznamů. Jelikož úkol má stejné vlastnosti jako krok a pouze jej obohacuje o další vlastnosti, je navržena jeho třída jako potomek třídy pro krok, TStep. Zároveň také může úkol obsahovat neomezené množství kroků, přičemž každý krok může patřit pouze jednomu úkolu.

1.11.3.2Ovládání a spravování dat

Mezi ostatní třídy patří „GUI“, „Database“, „Settings“ a „State“. Každá tato třída se stará o nějakou oblast celé aplikace. Při jejich implementaci je tedy vhodné využít návrhový vzor „singleton“ (jedináček), který zajistí, že z každé třídy bude možné vytvořit pouze

jednu instanci. Třída Database se stará o práci s databází indexedDB, konkrétně umožňuje vkládat, upravovat, mazat a načítat data z databáze. Třída Database také umožňuje převod dat do formátu JSON, aby je bylo možné jednoduše stáhnout a přenášet. Třída Settings obsahuje základní uživatelské nastavení aplikace a ukládá je do uložiště Local Storage. Třída GUI obstarává všechnu práci s grafikou aplikace, stará se tedy zejména o upravování DOM modelu. Třída State zpracovává a uchovává stav aplikace, také propojuje funkce ostatních tříd aplikace, obsahuje všechnu základní logiku a spouští inicializaci aplikace.



Obrázek 12 – Doménový model struktury kódu demonstrační aplikace (Vlastní zpracování)

1.12 Dokumentace tvorby programu v jazyce CoffeeScript

Tato kapitola obsahuje dokumentaci tvorby demonstrační aplikace v jazyce CoffeeScript. První část kapitoly je věnována samotné instalaci potřebných programů pro vývoj, dále jsou popsány použité knihovny třetích stran a poté jsou rozebrány části zdrojového kódu. Ukázky kódu obsahují pouze nejdůležitější části kódu, nikoliv celý zdrojový kód.

1.12.1 Instalace a tvorba projektu

Tvorba projektu je zde velmi jednoduchá, stačí mít libovolný textový editor určený pro programování a v něm nainstalované rozšíření s kompilátorem pro předklad CoffeeScriptu do JavaScriptu.

1.12.2 Knihovny

Pro jednodušší implementaci budou při vývoji aplikace v CoffeeScriptu využity knihovny „jQuery“ a „db.js“. jQuery je populární JavaScriptová knihovna pro interakci mezi JavaScriptem a HTML a db.js je knihovna zjednodušující práci s databází indexedDB, která je bez použití takovéto knihovny značně složitější.

1.12.3 Uživatelská data

Jak již bylo zmíněno pro zpracování uživatelských dat jsou navrhnuty třídy TStep, Todo a TList. Nejjednodušší třídou je právě třída pro krok úkolu, tedy TStep. Aby byla třída viditelná mezi soubory je nutné v CoffeeScriptu ji vytvořit jako součást globálního objektu window. V tomto případě je výhodné pro vlastnosti objektu využít zápis v konstruktoru se znakem @, neboť tento zápis jednak vytvoří vlastnost s tímto jménem v objektu, ale také ji při tvorbě instance předá hodnotu z konstruktoru, bez toho, aby tuto logiku musel vývojář sám zapsat do těla konstruktoru. Všem vlastnostem v objektu je také přiřazená původní hodnota. Mimo konstruktor obsahuje třída také dvě metody pro práci s databází, tyto metody obsahují parametr „callback“, pro kód, který předávají třídě databáze pracující s asynchronní databází indexedDB, aby se vykonal až po dokončení asynchronní operace s databází.

```
class window.TStep
    constructor: (@Id = -1, @Text="", @ParentId = window.UndividedTListId,
      @Done = false) ->
    Update: (callback) ->
      window.Database.UpdateTStep this, callback
    Remove: (callback) ->
      window.Database.RemoveTStep this.Id, callback
```

Podobnou třídou je třída Todo, která dědí z třídy TStep pomocí klíčového slova „extends“. Z tohoto důvodu také předává třída v konstruktoru parametry konstruktoru třídy TStep pomocí funkce „super“.

```
class window.Todo extends window.TStep
    constructor: (@Id = -1, @Text="", @ParentId = window.UndividedTListId,
      @Done = false, @Important = false,
      @CreateDate = new Date().toDateString(),
      @EndDate=window.DateMaxValue.toDateString(), Note="") ->
      super()
    Update: (callback) ->
      window.Database.UpdateTodo this, callback
    Remove: (callback) ->
      window.Database.RemoveTodo this.Id, callback
    GetTodoSteps: (callback) ->
      window.Database.GetTStepsByParentId this.Id, callback
    AddStep: (text, callback) ->
      window.Database.AddTStep text, this.Id, callback
```

1.12.4 Zpracování HTML

Pro práci s HTML je výhodné využít knihovnu jQuery. První ukázka níže obsahuje funkci pro dynamickou tvorbu HTML elementů dle uživatelských dat. Celý dynamický HTML obsah se tak nejdříve uloží do proměnné ve tvaru textového řetězce, ve které je využito pro zápis neřetězcových dat interpolace, tedy všechna data jsou uvedená v řetězci v blocích složených závorek začínajících znakem „#“ a tím převedena na součást řetězce. Poté je možné vložit tento HTML obsah do těla nějakého existujícího HTML elementu na stránce a přidat reakce na události nových HTML elementů, v ukázce například kliknutí na HTML element „li“.

```
CreateTList: (tList, lstId, container) ->
        tList.GetUnfinishedTodosCount((result) ->
            markup = $("
            <li id='#{lstId}' class='LeftSidebar-item
                  #{if window.State.CurrentTList == tList
                  then "is-selected"}'>
                <span class='LeftSidebar-itemIcon #{tList.Icon}'></span>
                <span class='LeftSidebar-itemText'>#{tList.Name}</span>
                <span class='LeftSidebar-itemCount'>#{result}</span>
            </li>")
            markup.appendTo "##{container}"
            $("##{lstId}").click () -> window.State.SwitchTList tList)
```

Knihovnu jQuery lze využít také na jednoduché přidávání a odstraňování CSS tříd HTML elementů, jako na následující ukázce.

```
ShowSearchView: ->
        $("#lblHeaderTListIcon").addClass "u-d-none"
        $("#txtHeaderTListName").addClass "u-d-none"
        $("#SearchViewEditor").removeClass "u-d-none"
```

1.12.5 Spravování dat

Jak již bylo zmíněno pro spravování dat je v demonstrační aplikaci využito uložišť indexedDB, pro všechna data týkající se úkolů a Local Storage, pro ukládání nastavení aplikace. Nastavení aplikace, tak zpracovává třída Settings. Tato třída pro ukázku obsahuje pouze jednu vlastnost pro nastavení, a to je barevný mód aplikace. Třída Settings je stejně jako třídy GUI, Database a State navrhnuta jako jedináček, v CoffeeScriptu je pro tento návrhový vzor doporučena implementace vyobrazená v následující ukázce, kde tato třída obsahuje v těle další soukromou třídu, zpřístupňující pouze pomocí své statické metody „Current“. Tuto metodu je nutné přidat do objektu window, aby bylo možné k tomuto objektu přistupovat ze všech souborů. S uložištěm Local Storage lze v CoffeeScriptu pracovat stejně jako v JavaScriptu pomocí objektu „localStorage“.

```
class Settings
  settings = null
  class PrivateClass
    SetDarkMode: (value) ->
      localStorage.setItem "DarkMode", value
    GetDarkMode: ->
        darkMode = localStorage.getItem "DarkMode"
        if darkMode is null then false;
        darkMode
  @Current: ->
    settings ?= new PrivateClass()
window.Settings = Settings.Current()
```

Pro práci s databází indexedDB je pro zjednodušení v demonstrační aplikace využito knihovny db.js. Pro práci s tímto uložištěm je navrhnuta v aplikaci třída Database. Na ukázce níže je metoda pro získání všech seznamů úkolů z databáze. Jelikož je komunikace s databází asynchronní, metoda obsahuje parametr pro kód, který se vykoná po dokončení komunikace s databází. Komunikace s databází je dokončena v těle funkce „then“, kde jsou získaná data převedena na objekty třídy TList a poté vykonán kód parametru callback, pokud byl zadán.

```
GetAllTLists: (callback) ->
      database.Server.TodoLists.query()
        .all()
        .execute()
        .then (results) ->
          tlists = []
          for item in results
            tlist = new window.TList(item.id, item.name, item.icon,
              item.unfinishedOpened, item.finishedOpened, item.orderTodo)
            tlists.push(tlist)
          if callback isnt undefined then callback(tlists) else tlists
```

Poslední ukázkou práce s databází je metoda pro přidání nového seznamu úkolů do databáze. Data objektu třídy TList se nejdříve převedou pro databázi do tradičního JSON formátů s malými písmeny na začátku vlastností a následně jsou uložena do databáze. Po dokončení operace funkce provede kód v parametru callback.

```
AddTlist: (newName, callback) ->
      database.Server.TodoLists.add
        name: newName
        icon: ""
        unfinishedOpened: true
        finishedOpened: false
        orderTodo: window.OrderTodo.None
      .then () ->
        if callback isnt undefined then callback()
```

1.13 Dokumentace tvorby programu v jazyce TypeScript

Tato kapitola dokumentuje tvorbu demonstrační aplikace v jazyce TypeScript. Kapitola je obsahem a ukázkami velmi podobná předešlé kapitole.

1.13.1 Instalace

Tvorba aplikace v TypeScriptu je opět velmi jednoduchá, pro vývoj stačí mít nainstalovaný libovolný vývojářský textový editor s rozšířením pro překlad TypeScriptu do JavaScriptu.

1.13.2 Knihovny

Stejně jako aplikace vyvíjené v jazyce CoffeeScript budou i zde použity knihovny jQuery a db.js.

1.13.3 Uživatelská data

Implementace tříd aplikace v TypeScriptu je velmi podobná jako v CoffeeScriptu, nemá však tak minimalistický zápis jako CoffeeScript a atributy třídy navíc obsahují datové typy a modifikátory přístupu.

```
class TStep {
    public Id: number;
    public Text: string;
    public ParentId: number;
    public Done: boolean;

public constructor(id: number = -1, text: string = "",
        parentId: number = UndividedTListId,
        done: boolean = false) {
        this.Id = id;
        this.Text = text;
        this.ParentId = parentId;
        this.Done = done;
    }
    public Update(callback): void {
        Database.Current().UpdateTStep(this, callback);
    }
    public Remove(callback): void {
        Database.Current().RemoveTStep(this.Id, callback);
    }
}
```

1.13.4 Zpracování HTML

Zpracování HTML elementů je totožné jako při použití CoffeeScriptu.

```
public CreateTList(tList:TList, lstId:string, container:string): void {
        tList.GetUnfinishedTodosCount((result) => {
            let markup: JQuery = $(`
            <li id='${lstId}' class='LeftSidebar-item
                ${State.Current().CurrentTList == tList ?
                "is-selected" : ""}'>
                <span class='LeftSidebar-itemIcon ${tList.Icon}'></span>
                <span class='LeftSidebar-itemText'>${tList.Name}</span>
                <span class='LeftSidebar-itemCount'>${result}</span>
            </li>
            `);
            markup.appendTo(`#${container}`);
            $(`#${lstId}`).click(() => { State.Current().SwitchTList(tList);
            });
        });
    }
```

1.13.5 Spravování dat

Implementace návrhového vzoru jedináček je v TypeScriptu vytvářena stejně jako v ostatních třídně objektově orientovaných jazycích. Tedy konstruktor třídy obsahuje privátní modifikátor přístupu, aby nešlo zvenčí vytvořit novou instanci třídy. K jediné instanci třídy je tak možné přistupovat pouze skrz metodu třídy „Current“, která při prvním zavolání vytvoří novou instanci a při dalších vrací již vytvořenou instanci. Pro práci s Local Storage je opět využit objekt „localStorage“.

```
class Settings {
    private static settings: Settings;
    private constructor() { }

    public static Current(): Settings {
        if (!this.settings) {
            this.settings = new Settings();
        }
        return this.settings;
    }
    public SetDarkMode(value: string) {
        localStorage.setItem("DarkMode", value);
    }
    public GetDarkMode() {
        let darkMode: string = localStorage.getItem("DarkMode")
        return (darkMode == null) ? false : darkMode;
    }
}
```

Práce s databází indexedDB je také totožná s implementací v jazyce CoffeeScript.

```
public GetAllTLists(callback?): void {
        this.Server.TodoLists.query()
            .all()
            .execute()
            .then((results) => {
                let tlists: TList[] = []
                for (let item of results) {
                    let tlist: TList = new TList(item.id, item.name,
                       item.icon, item.unfinishedOpened,
                       item.finishedOpened, item.orderTodo);
                    tlists.push(tlist);
                }
                if (callback != undefined)
                    callback(tlists);
            });
    }
```

1.14 Dokumentace tvorby programu v jazyce Dart

Tato kapitola obsahuje dokumentaci tvorby demonstrační aplikace v jazyce Dart. V kapitole opět není ukázán celý zdrojový kód ale pouze nejvýznamnější části.

1.14.1 Instalace

Instalace potřebných nástrojů pro tvorbu aplikací v jazyce Dart je náročnější než v minulých jazycích. Nejprve je nutné nainstalovat vývojářskou sadu nástrojů „Dart SDK“. Výhodné je také nainstalovat vývojářský textový editor s rozšířením pro podporu jazyka Dart. Pro tvorbu, testování a sestavení aplikace je potřeba pracovat s nástroji jazyka Dart pomocí příkazového řádku.

1.14.2 Knihovny

Jazyk Dart obsahuje velké množství standardních knihoven, proto zde není nutné používat pro práci s HTML elementy JavaScriptovou knihovnu jQuery. Práce s databází indexedDB je však i zde složitá, a proto je pro ni využita JavaScriptová knihovna db.js společně s vlastním potřebným JavaScriptovým kódem, neboť komunikace s JavaScriptovými knihovnami v Dartu není tak jednoduché jako v jazycích TypeScript a CoffeeScript.

1.14.3 Uživatelská data

Implementace tříd pro uživatelská data je téměř totožná jako v jazyce TypeScript, neboť je zápis velmi podobný a atributy také obsahují datové typy a modifikátory přístupu, které jsou však zde určovány dle zápisu názvu atributu.

```
import 'Database.dart';
class TStep {
  int Id;
  String Text;
  int ParentId;
  bool Done;

  TStep({int id = -1, String text = '', int parentId = UndividedTListId,
      bool done = false}) {
    Id = id;
    Text = text;
    ParentId = parentId;
    Done = done;
  }
  void Update(Function callback) {
    Database.Current.UpdateTStep(this, callback: callback);
  }
  void Remove(Function callback) {
    Database.Current.RemoveTStep(Id, callback: callback);
  }
}
```

1.14.4 Zpracování HTML

Pro práci s HTML elementy je vhodné v jazyce Dart využít standardní knihovnu „dart:html“, se kterou se pracuje podobným způsobem jako s JavaScriptovou knihovnou jQuery. V jazyce Dart je pro dynamickou tvorbu elementů namísto textového řetězcem, jako v TypeScriptu, či CoffeeScriptu vhodnější pracovat s jednotlivými HTML elementy jako s objekty. Proto v následující ukázce je ke každému HTML elementu vytvořen v jazyce Dart objekt, kterému jsou následně nastaveny atributy a je vložen do HTML struktury aplikace. Při nastavování atributů objektů lze využít zkrácený zápis pomocí dvou teček. K elementům v HTML struktuře je možné přistupovat pomocí funkce „querySelector“.

```
void CreateTList(TList tList, String lstId, String container, String result)
{
   var li = LIElement()
     ..id = lstId
     ..onClick.listen((e) {
       State.Current.SwitchTList(tList);
     })
     ..classes.add('LeftSidebar-item');
   if (State.Current.CurrentTList == tList) {
     li.classes.add('is-selected');
   }
   var spanIcon = SpanElement()
     ..className = 'LeftSidebar-itemIcon ' + tList.Icon;
   li.children.add(spanIcon);
   var spanText = SpanElement()
     ..classes.add('LeftSidebar-itemText')
     ..text = tList.Name;
   li.children.add(spanText);
   var spanCount = SpanElement()
     ..classes.add('LeftSidebar-itemCount')
     ..text = result;
   li.children.add(spanCount);
   querySelector('#${container}').children.add(li);
}
```

Přístup k elementům v HTML struktuře je podrobněji zobrazen na ukázce níže, kde jsou nastavovány jednotlivým elementům CSS třídy.

```
void ShowSearchView() {
    querySelector('#lblHeaderTListIcon').classes.add('u-d-none');
    querySelector('#lblHeaderTListIcon').classes.add('u-d-none');
    querySelector('#SearchViewEditor').classes.remove('u-d-none');
}
```

1.14.5 Spravování dat

Implementace návrhového vzoru jedináček je opět velmi podobná jako v ostatních třídně objektově orientovaných jazycích. Instance je zde však přístupná pomocí statické konstanty „Current“ a konstruktor třídy obsahuje klíčové slovo „factory“. Takto označený konstruktor umožňuje vytvořit instanci třídy pouze jednou a uložit ji do paměti pro další volání.

```
import 'dart:html';

class Settings {
  static final Settings Current = Settings._settings();
  factory Settings() {
    return Current;
  }
  Settings._settings();

  void SetDarkMode(bool value) {
    window.localStorage['DarkMode'] = value.toString();
  }
  String GetDarkMode() {
    var darkMode = window.localStorage['DarkMode'];
    return (darkMode == null) ? 'false' : darkMode;
  }
}
```

Kvůli složitosti s komunikací s databázi indexedDB je komunikace s ní zde realizována pomocí JavaScriptu. JavaScriptu je v Dartu nutné používat pomocí přerušeních definovaných v knihovně „dart:js“. Komunikace s JavaScriptem je vyobrazena na následující ukázce kódu.

```
js.context["Database"].callMethod('InitializeDatabase');
```

1.15 Dokumentace tvorby programu v Blazoru

Poslední dokumentace je věnována demonstrační aplikaci vytvořené ve frameworku Blazor a jazyku C#.

1.15.1 Instalace

Framework Blazor s jazykem C# je nejjednodušší používat pomocí vývojového integrovaného prostředí „Visual Studio“. Tento program obsahuje všechna potřebná rozšíření a je optimalizovaný pro vývoj aplikací ve frameworku Blazor a jazyce C#.

1.15.2 Knihovny

Jazyk C# je součástí .NET frameworku, který obsahuje velké množství standardních knihoven. Mimo těchto knihoven je však vhodné využít i knihovny třetích stran pro práci s uložišti. Pro práci s indexedDB je využita knihovna TG.Blazor.IndexedDB, bez které by bylo nutné vytvořit celou komunikaci v JavaScriptu stejně jako v aplikaci vytvořené v jazyce Dart a poté tento JavaScriptový kód volat pomocí přerušení. Stejně tak je i využita knihovna Blazored.LocalStorage využitá pro práci s uložištěm Local Storage.

1.15.3 Uživatelská data

Implementace tříd pro uživatelská data je také podobná ostatním variantám. V metodách pro práci s databází však není pro správné pořadí vykonávání kódu využito parametrů „callback“, ale používá se klíčové slovo „await“, díky kterému se následující kód začne vykonávat až po dokončení zavolané asynchronní funkce.

```
public class TStep {
    public long? Id { get; set; }
    public long? ParentId { get; set; } = Database.UndividedTListId;
    public bool Done { get; set; } = false;
    public string Text { get; set; }

    public TStep(string text, long parentId = Database.UndividedTListId) {
        ParentId = parentId;
        Text = text;
    }
    public virtual async Task Update() {
        await Database.Current.UpdateTStep(this);
    }
    public virtual async Task Remove() {
        await Database.Current.RemoveTStep(Id.Value);
    }
}
```

1.15.4 Zpracování HTML

Pro práci s HTML je nutné využít „razor engine“, který umožňuje zapisovat kód jazyka C# přímo do HTML struktury. Komponenta pro dynamické vytváření seznamů úkolů dle uživatelských dat je na následující ukázce kódu.

```
<li class="LeftSidebar-item" @onclick="(_) => Click(TodoList)">
    <span class="LeftSidebar-itemIcon @TodoList.Icon"></span>
    <span class="LeftSidebar-itemText">@TodoList.Name</span>
    <span class="LeftSidebar-itemCount">@UnfinishedTasksCount</span>
</li>
@code {
    [Parameter] public TList TodoList { get; set; }
    [Parameter] public Func<TList, Task> Click { get; set; }
    public string UnfinishedTasksCount { get; set; }

    protected override async Task OnParametersSetAsync() {
        await base.OnParametersSetAsync();
        UnfinishedTasksCount = await TodoList.GetUnfinishedTodosCount();
    }
}
```

1.15.5 Spravování dat

Na následující ukázce je implementace návrhového vzoru jedináček pro třídu Settings, stejně jako v ostatních variantách. Návrhový vzor je implementován podobným způsobem jako v TypeScript a pro ukládání dat do Local Storage je využita komunitní knihovna Blazored.LocalStorage.

```
public class Settings {
    private static Settings settings;
    public static Settings Current {
        get {
            if (settings == null) settings = new Settings();
            return settings;
        }
    }
     private Settings() { }
     private bool? darkMode;
     public bool? DarkMode {
         get => darkMode;
         set {
             darkMode = value;
             LocalStorage.SetItem(nameof(DarkMode), value);
             NotifyPropertyChanged();
         }
     }
}
```

Poslední ukázky jsou opět věnovány ukládání dat do databáze indexedDB pomocí třídy Database. Zde je ukládání dat velmi jednoduché díky knihovně TG.Blazor.IndexedDB a také díky tomu, že je možné využívat jednoduše klíčové slovo await pro počkání na dokončení asynchronní funkce.

```
public async Task<List<TList>> GetAllTLists() =>
            await DbManager.GetRecords<TList>(TListDbName);
```

Ukládání dat do databáze je zde také velmi jednoduché díky použité knihovně.

```
public async Task AddTList(string name)
{
    var newRecord = new StoreRecord<TList>
    {
        Storename = TListDbName,
        Data = new TList(name)
    };
    await DbManager.AddRecord(newRecord);
}
```

1.16 Porovnání a zhodnocení alternativ z pohledu uživatele

Využitá platforma, knihovny a jazyk pro aplikaci mají vliv jak na samotného vývojáře, tak i na uživatele, samozřejmě na obě skupiny mají vliv odlišným způsobem. Vývojáři se spíše při výběru technologií zaměří na struktury, přenositelnost kódu a další podobné faktory, zatímco uživatelé vnímají u aplikací zejména výkon, datovou náročnost a výskyt běhových chyb. Výsledné údaje byly zjištěny zejména z jednotlivých implementací demonstračních aplikací z předešlých kapitol. Výsledky však nejsou zcela přesné a objektivní, neboť jsou závislé na způsobu vývoje, znalostech a dalších faktorech.

1.16.1 Datová náročnost

Datovou náročností je myšlena velikost všech stažených komprimovaných dat při spuštění aplikace.

Tabulka 5 – Porovnání datové náročnosti alternativ (Vlastní zpracování)

	Kód [kB]	Knihovny [kB]	Struktura a grafika [kB]	Celková velikost [kB]	Hodnocení
Blazor	90,77	3	142,348	236,118	3
CoffeeScript	14,522	35,2	142,348	192,07	1
TypeScript	13,525	35,2	142,348	191,073	1
Dart	53,3	3,9	142,348	199,548	2

Demonstrační aplikace napsané v TypeScriptu, CoffeeScriptu a Dartu jsou téměř stejně datově náročné, tedy stažená velikost při vstupu do těchto aplikací se pohybuje v rozmezí od 190 kB do 200 kB. To je však způsobeno tím, že pro vývoj v jazycích TypeScript a CoffeeScript byly využity knihovny pro práci s HTML elementy a pro práci s databází, jinak by jejich aplikace byly o něco menší než aplikace napsaná v Dartu, neboť Dart je kvůli většímu množství standardních knihoven datově náročnější. Nejvíce datově náročnou variantou je však aplikace vytvořená ve frameworku Blazor s přibližnou velikostí 236 kB. To je způsobeno tím, že Blazor je kompletní framework pro vývoj webových aplikací na rozdíl od ostatních alternativ a také obsahuje velké množství standardních knihoven .NET frameworku. Velikostní rozdíly mezi projekty jsou však, i přes to poměrně nízké.

1.16.2 Načítací doba

V této kategorii se porovnává čas uběhnutý při spuštění aplikace, než je stránka celá připravena a uživatel může aplikaci využívat.

Tabulka 6 – Porovnání načítací doby alternativ (Vlastní zpracování)

	1. [ms]	2. [ms]	3. [ms]	4. [ms]	5. [ms]	6. [ms]	7. [ms]	8. [ms]	9. [ms]	10. [ms]	Průměr [ms]	Hodnocení
Blazor	1130	1220	1170	1130	1140	1140	1450	1200	1350	1140	1207	4
CoffeeScript	430	381	508	483	413	397	306	382	496	375	417,1	1
TypeScript	368	617	447	385	457	408	348	299	330	378	403,7	1
Dart	415	341	327	482	342	414	398	361	396	363	383,9	1

Opět transpilované varianty mají podobné výsledky a jejich načítací doba je průměrně 400 ms. Aplikace v Blazoru se načítala třikrát delší dobu, tedy průměrně 1200 ms, jedná se tudíž o významný rozdíl.

1.16.3 Podpora

Jedinými podporovanými jazyky pro běh v prohlížečích je JavaScript a binární formát WASM.

Tabulka 7 – Porovnání alternativ, dle podpory v prohlížečích (Vlastní zpracování)

	Prohlížeče	Hodnocení
Blazor	92,54%	2
CoffeeScript	100%	1
TypeScript	100%	1
Dart	100%	1

Jelikož je WebAssembly značně novější technologií není ještě podporovaný všemi prohlížeči, na rozdíl od JavaScriptu. V tomto mají tedy opět výhodu jazyky transpilované do JavaScriptu. WebAssembly je však již podporovaný ve většině nejpoužívanějších prohlížeč. Mezi prohlížeče, které dosud WebAssembly nepodporují patří zejména méně používané prohlížeče pro mobilní zařízení, či prohlížeče, které jsou na konci podpory, jako je například Internet Explorer.

1.16.4 Výkon

Výkon, tedy konkrétně rychlost vykonávání kódu jednotlivých alternativ v prohlížeči byla měřena na pěti odlišných malých úlohách. Rychlost alternativ byla měřena za stejných podmínek v prohlížeči „Chrome“.

Tabulka 8 – Porovnání výkonu alternativ (Vlastní zpracování)

	Bubble sort [ms]	Caesarova šifra [ms]	JSON serializace [ms]	JSON deserializace [ms]	Generování HTML [ms]	Hodnocení
Blazor	10,128	21,31	3,02	4,54	5,383	5
CoffeeScript	2,1	0,5	0,1	0,1	2,9	1
TypeScript	1,1	0,2	0,1	0,1	2,7	1
Dart	0,6	0,1	1,7	1,5	18,7	3
Firefox Blazor	6,3	10,9	3,4	5,4	2,7	4

První úlohou byl pomalý jednoduchý třídicí algoritmus „Bubblesort“, konkrétně bylo cílem seřadit vzestupně soubor čísel o 400 číslech. Úlohu řešil nejrychleji jazyk Dart s průměrným časem 0,6 ms, dále TypeScript s průměrným časem 1,1 ms, CoffeeScript s časem 2,1ms a poslední byl Blazor s podstatně vyšším průměrným časem 10,1 ms.

Další úlohou bylo zašifrovat textový řetězec o velikost 500 slov pomocí Caesarovy šifry. Zde byly výsledky velmi podobné, zatímco transpilované varianty řešili úlohu v čase řádu stovek mikrosekund, Blazor řešil úlohu průměrně 21 ms.

Třetí úloha byla věnována převodu velkého množství dat do formátu JSON a čtvrtá úloha zpět z formátu JSON do pole objektů v daném jazyce. Zde byly nejrychlejšími variantami TypeScript a CoffeeScript využívající JavaScriptové funkce pro práci s formátem JSON, řešili úlohu v řádu stovek mikrosekund. O něco pomaleji řešil úlohu jazyk Dart, průměrně za 1,5 ms a nejdéle trvalo vyřešit úlohu programu vytvořeném v Blazoru, průměrně 5 ms.

Poslední úlohou bylo dynamické generování HTML obsahu. Konkrétně se jednalo o generování tabulky násobků dvou čísel. Tuto úlohu řešil nejrychleji program napsaný

v TypeScriptu, průměrně za 2,7 ms, poté CoffeeScript za 2,9ms, dále Blazor za 5,4 ms a poslední se značně vyšším časem byl jazyk Dart, průměrně za 18,7 ms.

I přes to, že WebAssembly je teoreticky mnohem rychlejší než JavaScript, JavaScript je stále ve spouště oblastí, jako třeba ve tvorbě webových aplikací výkonnější. Určitě výsledek neznamená, že je JavaScript vždy rychlejší, existuje například spousta her na internetu vytvořených pomocí WebAssembly, které by v JavaScriptu takto výkonné nikdy nebyly. Hlavním důvodem, proč je JavaScript rychlejší je opět jeho mnohem delší vývoj, než má WebAssembly. Zpracování jazyka JavaScript v prohlížečích je v současné době dobře optimalizované a velmi výkonné, zatímco zpracování WebAssembly na takto dobré úrovni v prohlížečích není. Dá se tedy očekávat, že výkon WebAssembly v prohlížečích ještě značně poroste. Důkazem může být výkon WebAssembly v prohlížeči Firefox, který je ve zpracování WebAssembly velmi pokročilý a všechny provedené úlohy byly zde v projektu vytvořeném v Blazoru téměř dvakrát rychlejší než v prohlížeči Chrome, čímž se například v úloze s generováním HTML dostal na stejný čas jako TypeScript.

1.16.5 Četnost běhových chyb

V této kategorii se porovnává počet chyb vzniklých při běhu aplikace. Samotné měření bylo realizováno počítáním neočekávaných chyb vzniklých při testování hotových částí aplikace. Jedná se zde však o částečně subjektivní hodnocení, neboť je závislé na schopnostech vývojáře.

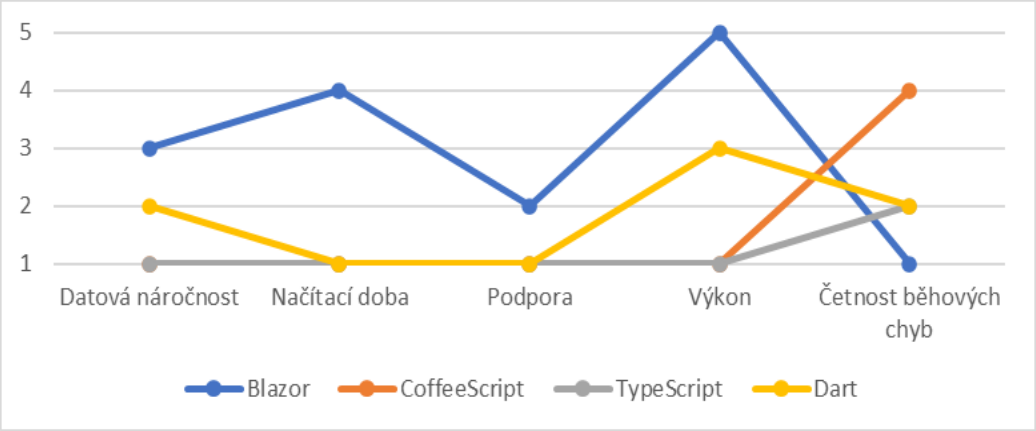
Tabulka 9 – Porovnání četnosti běhových chyb alternativ (Vlastní zpracování)

	Počet chyb	Hodnocení
Blazor	6	1
CoffeeScript	22	4
TypeScript	9	2
Dart	11	2

Díky dobré struktuře, hlídání statických typů a dalším funkcionalitám má však Blazor s jazykem C# nejmenší pravděpodobnost vzniku běhových chyb. Podobně jsou na tom i jazyky TypeScript a Dart. Nejhorší variantou, je tak v tomto ohledu jazyk CoffeeScript.

1.16.6 Výsledné zhodnocení

Výsledky z jednotlivých kategorií jsou graficky zobrazeny na grafu na obrázku 14.



Obrázek 13 – Graf porovnání alternativ z pohledu uživatele (Vlastní zpracování)

Dle grafu je možné usoudit, že v současné době je pro uživatele vhodnější s ohledem na načítací dobu, podporu prohlížečů a výkon vhodnější používat alternativy, které se transpilují do JavaScriptu. Na druhou stranu ve všech těchto třech kategoriích lze očekávat značná zlepšení, zejména u výkonu. Pro malé aplikace s ohledem na datovou náročnost může být vhodnější použít TypeScript, či CoffeeScript, neboť Dart a Blazor jsou o něco datově náročnější z důvodu jejich standardních knihoven a dalším doplňujícím funkcionalitám. I přes významné výhody v současné době u jazyků překládajících se do JavaScriptu, může být často výhodnější využít jazyk s překladem do formátu WASM, a to z důvodu některých výhod formátu WASM, ale i výhod samotného jazyka.

1.17 Porovnání a zhodnocení alternativ z pohledu vývojáře

Toto hodnocení je zaměřeno na vývojáře, porovnávají si tedy spíše vlastnosti kódu, dostupnost knihoven a další podobné faktory.

1.17.1 Struktura kódu

V této kategorii jsou jazyky hodnoceny dle možností pro efektivní strukturaci kódu. Hodnotí se tedy funkcionality pro dělení kódu jako jsou například jmenné prostory, možnosti pro objektově orientovaný přístup, možnosti pro využití klasických návrhových vzorů a podobné vlastnosti.

Tabulka 10 – Porovnání struktury kódu alternativ (Vlastní zpracování)

	Hodnocení
Blazor	1
CoffeeScript	4
TypeScript	2
Dart	2

Nejlepším jazykem je v tomto zhodnocení jazyk C#, podobně je na tom i jazyk TypeScript, který přebírá spoustu funkcionalit z jazyka C# a nabízí tak velmi podobné vlastnosti, je však omezován vlastnostmi jazyka JavaScript. Podobné možnosti nabízí také jazyk Dart, ten však k některým funkcionalitám přistupuje jiným vlastním způsobem, nebo je vůbec neobsahuje. Na druhou stranu Dart, také přidává některé nové funkcionality navíc oproti ostatním jazykům. Nejméně možností pro strukturaci nabízí jazyk CoffeeScript. I v jazyce CoffeeScript však lze vytvářet například funkcionalitu jmenných prostorů, i když je přímo nepodporuje.

1.17.2 Vhodnost pro velké aplikace

Kategorie hodnotí zejména možnosti pro přehlednost kódu a modularizaci.

Tabulka 11 – Porovnání alternativ, dle vhodnosti pro velké aplikace (Vlastní zpracování)

	Vhodné pro velké aplikace	Hodnocení
Blazor	ano	1
CoffeeScript	ne	5
TypeScript	ano	1
Dart	ano	1

Zhodnocení jazyků v této kategorii je velmi podobné jako v kategorii minulé, tedy jazyk C# je nejvíce vhodný pro vývoj velkých aplikací, podobně je na tom jazyk TypeScript, který byl dokonce za účelem udělat jazyka JavaScript vhodný pro velké aplikace vytvořen, dále jazyk Dart a nejméně vhodným jazykem v kategorii je CoffeeScript. Jazyk CoffeeScript může být dokonce pro vývojáře méně přehledný než JavaScript kvůli absenci složených závorek pro vymezení prostoru bloku. CoffeeScript tedy na rozdíl od ostatních jazyků není ani vhodný pro vývoj velkých aplikací.

1.17.3 Statický typový systém

V této kategorii se hodnotí, zda jazyk umožňuje zadávat proměnným statické datové typy. Dynamické typy umožňují všechny hodnocené jazyky, jejich nevýhodou je však, že pomocné vývojářské nástroje nedokážou k nim vytvářet nápovědy a dostatečně je kontrolovat. To je způsobena zejména tím, že je možné u dynamických proměnných jednoduchým přiřazením hodnoty možné změnit jejich datový typ. Naopak proměnné se statickými datovými typy je podstatně jednodušší kontrolovat, tedy kontrolovat operace prováděné s nimi a také je jednoduché k nim nabízet nápovědu.

Tabulka 12 – Porovnání alternativ, dle implementace statického typového systému (Vlastní zpracování)

	Statický typový systém	Hodnocení
Blazor	ano	1
CoffeeScript	ne	5
TypeScript	ano	1
Dart	ano	1

Statické datové typy tedy umožňují pouze jazyky C#, TypeScript a Dart. Existují však komunitní rozšíření pro CoffeeScript, přidávající podporu statických datových typů.

1.17.4 Možnost využití JavaScriptového kódu

Možnost alternativních jazyků využít kód napsaný v jazyce JavaScript je velmi důležité, neboť JavaScript byl dlouhou dobu jediným podporovaným jazykem pro běh v prohlížečích, a tak pro něj existuje velké množství knihoven, frameworků a hotových řešení. Také spousta vývojářů, když přechází na nějakou alternativu potřebuje, aby tato alternativa umožňovala používat kód, který již dříve napsali v JavaScriptu.

Tabulka 13 – Porovnání alternativ, dle složitosti využití JavaScriptového kódu (Vlastní zpracování)

	Složitost používání JavaScriptu	Hodnocení
Blazor	vyšší	3
CoffeeScript	nižší	1
TypeScript	nižší	1
Dart	vyšší	3

JavaScript je možné využívat ve všech variantách, nejjednodušší je však využívat v jazycích CoffeeScript a TypeScript. V případě CoffeeScriptu může vývojář využívat JavaScriptový kód bez jakýchkoliv problémů, prakticky lze k němu přistupovat jako by byl napsaný také v CoffeeScriptu. TypeScript již doporučuje používat pro JavaScriptový kód definiční soubory, kde jsou popsané jednotlivé funkce a další prvky kódu společně s datovými typy, vytvářet definiční soubory však není povinné. Poněkud složitější způsob pro používání kódu v JavaScriptu nabízí varianty Dart a Blazor. V těchto variantách je nutné k JavaScriptu přistupovat pomocí funkcí pro přerušení. V těchto variantách tedy není tak pohodlné a výhodné JavaScriptový kód využívat.

1.17.5 Komunita

Kategorie komunita, tedy velikost komunity programovacího jazyka je hodnocena zejména z důvodu, že velmi souvisí s dostupným množstvím komunitních knihoven, hotových

řešení, frameworků a studijních materiálů. Tato kategorie byla hodnocena na základě informací ze tří odlišných měření, tedy podle poměru dotazů na daný jazyk proti dotazům na všechny jazyky v roce 2020 na fóru StackOverflow, dále podle celkového počtu dotazů na daný jazyk na fóru StackOverflow a podle počtu repositářů používajících daný jazyk na stránce GitHub.

Tabulka 14 – Porovnání alternativ, podle velikost komunity (StackOverflow.com, GitHub.com)

	StackOverflow.com - poměr dotazů v roce 2020 [%]	StackOverflow.com - celkový počet dotazů	GitHub.com - celkový počet repozitářů	Hodnocení
Blazor	0,20%	4896	12995	3
CoffeeScript	0,05%	9738	7998	4
TypeScript	1,80%	140012	188055	1
Dart	1,00%	45391	38591	2

Výsledky ze všech tří měření byly velmi podobné, tedy největší komunita má jazyk TypeScript. Podstatně menší komunitu již má jazyk Dart. Blazor má malou komunitu zejména z důvodu, že se jedná o nejnovější technologii a taky může být na zkoumaných stránkách z důvodu, že se jedná o framework označený pouze jako projekt, či dotaz na jazyk C#. V současné době má tak nejmenší komunitu jazyk CoffeeScript.

1.17.6 Standardní knihovny

Dostatečné množství kvalitních standardních knihoven je pro vývojáře velmi důležité. Standardní knihovny většinou obsahují optimalizované funkcionality pro různé problémy a pro různé platformy. Bez nich musejí vývojáři funkcionality hledat v knihovnách třetích stran, či je musí vytvořit sami.

Tabulka 15 – Porovnání alternativ, podle standardních knihoven (Vlastní zpracování)

	Hodnocení
Blazor	1
CoffeeScript	4
TypeScript	4
Dart	2

Největší množství kvalitních standardních knihoven má jazyk C#, který je součástí velkého ekosystému .NET frameworku. Velké množství standardních knihoven obsahuje i jazyk Dart. Situace je značně horší u jazyků TypeScript a CoffeeScript, které JavaScript neobohacují o standardní knihovny a jsou tak závislé na standardních knihovnách JavaScriptu.

1.17.7 Dostupné platformy

Dostupnými platformami se myslí, pro jak velké množství různých platforem je jazyk optimalizovaný.

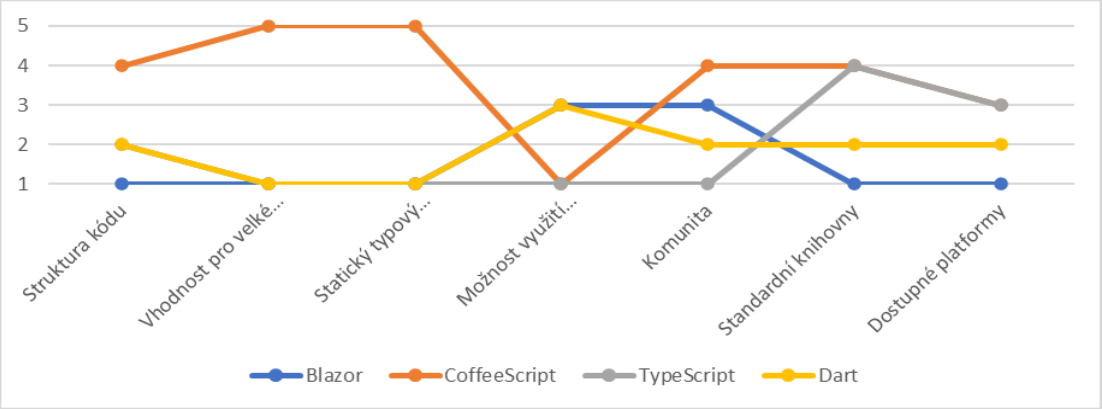
Tabulka 16 – Porovnání alternativ, podle dostupných platforem (Vlastní zpracování)

	Hodnocení
Blazor	1
CoffeeScript	3
TypeScript	3
Dart	2

Jazykem optimalizovaným pro největší množství platforem je opět C#. Jazyk C# je také možné využít pro velké množství projektů různých typů, a také lze mezi platformami často využívat naprosto stejný kód. Poměrně velké množství dostupných platforem nabízí i jazyk Dart, zejména díky frameworku Flutter. O něco horší je optimalizace pro různé platformy u jazyků TypeScript a CoffeeScript. I tyto jazyky je však možné využívat v prohlížeči a na serverech a pomocí dalších nástrojů i na počítačích a telefonech.

1.17.8 Výsledné zhodnocení

Výsledky porovnání z pohledu vývojáře je možné vidět na grafu na obrázku 15.



Obrázek 14 - Graf porovnání alternativ z pohledu vývojáře (Vlastní zpracování)

Porovnání z pohledu vývojáře je složitější, neboť je nutné porovnat více faktorů a také má větší tendenci být subjektivní, neboť může být silně ovlivněno znalostmi vývojáře. Toto hodnocení by také mohlo být nespravedlivé, neboť Blazor není programovací jazyk, ale framework pro jazyk C#. Bez něho však není možné spouštět v prohlížeči kód v jazyce C#. Je proto nutné se vyhnout zvýhodňování Blazoru kvůli jeho vlastnostem, které by ostatní alternativy také měli, kdyby byly využity společně s nějakým JavaScriptovým framework.

Dle grafu je nejvhodnější alternativou pro vývojáře jazyk C# s frameworkem Blazor, neboť se jedná o velmi pokročilý jazyk se silnou komunitou, velkým počtem kvalitních standardních knihoven a je vhodný pro velké množství platforem a projektů. Podobně výhodnými alternativami jsou TypeScript a Dart. TypeScript přebírá spousty vlastností z jazyka C#, a tak nabízí podobné funkcionality, těší se také velké popularitě, díky tomu, že se jedná o nadmnožinu JavaScriptu, která jej pouze obohacuje. Dart je také velmi pokročilým jazykem, v poslední době se však využívá spíše pro mobilní zařízení než pro skriptování v prohlížeči. Nejhorší alternativou, je tak v současné době CoffeeScript. CoffeeScript byl jednou z prvních alternativ JavaScriptu a na počátku se těšil velké popularitě. Dnes je už však zastíněn ostatními alternativami, neboť neobsahuje takové množství funkcionalit a není tak pokročilý jako ostatní alternativy.

Závěr

V bakalářské práci byly nalezeny, popsány a zhodnoceny dostupné alternativy jazyka JavaScript pro klientské zpracování dat ve webovém prostředí. Vybrané alternativy byly hodnoceny na základě výsledků vývoje malé demonstrační aplikace a byly zhodnoceny z pohledu vývojáře i z pohledu uživatele.

První polovina teoretické části práce byla věnována webovému prostředí. Zde byla popsána historie webu se zaměřením i na vývoj jazyků pro klientské zpracování dat, dále byly popsány webové prohlížeče a jednotlivé druhy obsahu ve webovém prostředí. Tyto informace tvoří naprostý základ pro další části práce. V této části práce byly následně popsány jednotlivé části webové aplikace a technologie potřebné pro vývoj webových aplikací. Popis technologií byl zaměřen zejména na klientské zpracování a ukládání dat. Byly zde také popsány technologie umožňující existenci alternativ jazyka JavaScript, konkrétně standard WebAssembly a proces transpilace.

Druhá polovina teoretické části popisuje samotný jazyk JavaScript, jeho vlastnosti, historii, výhody a nevýhody s odlišnostmi, kvůli kterým vznikají alternativy k tomuto jazyku. Dále byly popsány jednotlivé nejvýznamnější alternativy, tedy jazyky CoffeeScript, TypeScript, Dart a framework Blazor s jazykem C#. U každé této alternativy byly popsány základní vlastnosti, historický vývoj, výhody a nevýhody a základní syntaktické vlastnosti. Poslední část teoretické části je věnována dalším, často méně používaným, ale přesto významným alternativám jazyka JavaScript.

V praktické části byla vytvořena demonstrační aplikace v každé vybrané alternativě a následně porovnána s ostatními alternativami. Praktická část tedy začíná samotným návrhem demonstrační aplikace. Dále se věnuje uživatelskému rozhraní aplikace a návrhu struktury kódu, který je následně implementován jednotlivými alternativami. Proces tvorby aplikace byl u každé alternativy zdokumentován včetně instalace a potřebných knihoven pro vývoj. Hlavní část dokumentace byla věnována samotnému programování funkcionalit, kde významné části byly podrobněji popsány. Jazyky a demonstrační aplikace byly následně porovnány a zhodnoceny z pohledu uživatele i z pohledu tvůrce. Získané výsledky byly popsány a vyobrazeny v podobě neúplně ordinálně uspořádané množiny.

Dle výsledků získaných při hodnocení z pohledu uživatele je výhodnější v současné době pro vývoj webových aplikací využívat spíše jazyky, které se překládají do JavaScriptu, než ty, které se překládají do formátu standardu WebAssembly. Formát WebAssembly totiž v současné době nabízí menší podporu a nižší výkon pro webové aplikace než JavaScript. Formát WebAssembly je však znevýhodněn tím, že jeho vývoj je mnohem kratší než vývoj JavaScriptu, lze tak očekávat v budoucnu jeho zlepšení. Nejlepšími variantami z pohledu uživatele byly TypeScript a CoffeeScript, neboť Dart obsahuje velké množství standardních knihoven a je tak značně datově náročnější. Ze zhodnocení dle pohledu tvůrce vyšel jako nejlepší jazyk C# s jeho frameworkem Blazor, neboť se jedná o velmi pokročilý jazyk s velkým množstvím funkcionalit a knihoven. Jazyky TypeScript a Dart měly však v tomto pohledu také dobré zhodnocení. Nejhorší alternativou zde byl jazyk CoffeeScript, protože je mnohem méně pokročilý než ostatní alternativy a v současné době se jedná o velmi málo používaný jazyk.

Lze tak usoudit, že jazyky TypeScript, Dart a framework Blazor jsou velmi vhodnou náhradou jazyka JavaScript. Jazyk TypeScript je vhodný spíš pro vývojáře používající zejména JavaScript a alternativy Dart a Blazor jsou vhodnější pro vývojáře, které používají nějaký jazyk z rodiny jazyka C.

Seznam použitých zdrojů

FENTON, Steve (2017) Pro typescript. New York, NY: Springer Science+Business Media, ISBN 9781484232484.

HIMSHOOT, Peter (2019) Blazor revealed - building web applications in .NET. New York, NY, Springer Science+Business Media, ISBN 9781484243428.

MACCAW, Alex (2012) The little book on CoffeeScript. Sebastopol, CA: O'Reilly Media, ISBN 9781449321055.

ŽÁRA, Ondřej (2015) JavaScript: Programátorské techniky a webové technologie, Brno Computer Press, ISBN 9788025145739.

BROWN, Tiffany B., Kerry BUTTERS a Sandeep PANDA. *HTML5 okamžitě: [ovládněte HTML5 za víkend]*. Brno: Computer Press, 2014. ISBN 9788025142967.

LAZARIS, Louis. *CSS okamžitě*. Brno: Computer Press, 2014. ISBN 9788025141762.

NIEMCZYK, Kamil. Analýza jazyka Dart [online]. Ostava, 2015. Bakalářská práce. VŠB - Technická univerzita Ostava, Fakulta elektrotechniky a informatiky, Katedra informatiky.

PROCHÁZKA, Šimon. Vývoj webové aplikace [online]. Brno, 2019. Dostupné z: https://www.vutbr.cz/studenti/zav-prace/detail/118383.Bakalářská práce. Vysoké učení technické v Brně, Fakulta podnikatelská, Ústav informatiky. Vedoucí práce Petr Dydowicz.

BERNERS-LEE, Tim (1998) The World Wide Web: A very short personal history, http://www.w3.org/People/Berners-Lee/ShortHistory.html

The Weird History of JavaScript. In: Youtube [online]. 18.06.2019 [cit. 2021-01-04]. Dostupné z: https://www.youtube.com/watch?v=Sh6lK57Cuk4. Kanál uživatele Fireship.

CHAMPEON, Steve. JavaScript: How Did We Get Here? *O'Reilly Media Inc.* [online]. 4.6.2001 [cit. 2021-02-04]. Dostupné z: https://web.archive.org/web/20160719020828/http://archive.oreilly.com/pub/a/javascript/2001/04/06/js_history.html

DROZDÍK, Martin. WEBASSEMBLY A SMRT JAVASCRIPTU? *Bonsai Development* [online]. 3.9.2019 [cit. 2021-02-04]. Dostupné z: https://bonsai-development.cz/clanek/webassembly-a-smrt-javascriptu

LAMOLONY, Daniel. Static vs Dynamic websites - The pros and cons. *Lama Apps* [online]. 31.3.2019 [cit. 2021-02-04]. Dostupné z: https://www.lamaapps.com/blog/static-vs-dynamic-website-pros-cons/

SHUKLA, Vivek. [Here is simple...] In: Stackoverflow [online]. 30.6.2017 7:15 [cit. 2021-02-04]. Dostupné z: https://stackoverflow.com/questions/44840628/difference-between-website-and-web-application

Document Object Model. *MDN Web Docs* [online]. Mozilla, c2005-2021 [cit. 2021-02-04]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model

Introduction to the server side. *MDN Web Docs* [online]. Mozilla c2005-2021 [cit. 2021-02-05]. Dostupné z: https://developer.mozilla.org/en-US/docs/Learn/Server-side/First_steps/Introduction

Render trees. *Blazor University* [online]. [cit. 2021-02-04]. c2019-2021 Dostupné z: https://blazor-university.com/components/render-trees/

Difference Between Search Engine and Browser. *DifferenceBetween.net* [online]. 2011 [cit. 2021-02-04]. Dostupné z: http://www.differencebetween.net/technology/internet/difference-between-search-engine-and-browser/

HIWARALE, Uday. Pug.js to make your life easier with HTML templates. *Medium* [online]. 22.3.2018 [cit. 2021-02-05]. Dostupné z: https://medium.com/jspoint/pug-js-to-make-your-life-easier-with-html-templates-9c62273626e0

MICHÁLEK, Martin. Metodiky pro organizaci CSS kódu: Pište styly bez bolehlavů. *Vzhůru dolů* [online]. 21.7.2020 [cit. 2021-02-05]. Dostupné z: https://www.vzhurudolu.cz/prirucka/css-metodiky

MICHÁLEK, Martin. Průvodce CSS preprocesory: co a jak? *Vzhůru dolů* [online]. 10.3.2014 [cit. 2021-02-05]. Dostupné z: https://www.vzhurudolu.cz/blog/12-css-preprocesory-1

WETZEL, Ivo, Yi Jiang ZHANG a Tim RUFFLES. A Guide to JavaScript's Quirks and Flaws. *JavaScript Garden* [online]. [cit. 2021-02-06]. Dostupné z: https://bonsaiden.github.io/JavaScript-Garden/

ZACZEK, Staszek. How CoffeeScript Got Forgotten. *Medium* [online]. 26.1.2020 [cit. 2021-02-06]. Dostupné z: https://medium.com/better-programming/how-coffeescript-got-forgotten-812328225987

TypeScript. *Cleverism* [online]. [cit. 2021-02-06]. Dostupné z: https://www.cleverism.com/skills-and-tools/typescript/

BUCKETT, Chris. *Dart in Action*. Shelter Island (New York): Manning Publications Co., 2013. ISBN 9781617290862.

BOLTON, David. The Fall and Rise of Dart, Google's 'JavaScript Killer'. *Dice* [online]. 27.3.2019 [cit. 2021-02-07]. Dostupné z: https://insights.dice.com/2019/03/27/fall-rise-dart-google-javascript-killer/

Elm - A delightful language for reliable webapps. *Elm* [online]. [cit. 2021-02-07]. Dostupné z: https://elm-lang.org/

NANCE, Jared. TypeScript vs. JavaScript: Should You Migrate Your Project to TypeScript? *Stackify* [online]. 20.9.2017 [cit. 2021-02-07]. Dostupné z: https://stackify.com/typescript-vs-javascript-migrate/

REZVI, Mahdhi. JavaScript vs Dart: An Overview: A Comparison Between Two Popular Game Changers. *Bits and Pieces* [online]. 13.5.2020 [cit. 2021-02-07]. Dostupné z: https://blog.bitsrc.io/javascript-vs-dart-an-overview-e2879abde6a4

ČÁPKA, David. Lekce 1 - Úvod do C# a .NET frameworku. *ITnetwork.cz* [online]. [cit. 2021-02-08]. Dostupné z: https://www.itnetwork.cz/csharp/zaklady/c-sharp-tutorial-uvod-do-jazyka-a-dot-net-framework

JavaScript HTML DOM. *W3Schools* [online]. c1999-2021 [cit. 2021-02-09]. Dostupné z: https://www.w3schools.com/js/js_htmldom.asp

JAHODA, Bohumil. Javascriptové úložiště localStorage. *Zdroják* [online]. 26.4.2016 [cit. 2021-02-09]. Dostupné z: https://www.zdrojak.cz/clanky/javascriptove-uloziste-localstorage/

Stack Overflow Trends. *StackOverflow* [online]. 2021 [cit. 2021-02-09]. Dostupné z: https://insights.stackoverflow.com/trends?tags=blazor%2Ctypescript%2Cdart%2Ccoffeescript

KHARCHENKO, Natalia. 8 JavaScript Alternatives for Web Developers to Consider. *Codeburst* [online]. 17.1.2018 [cit. 2021-02-10]. Dostupné z: https://codeburst.io/8-javascript-alternatives-for-web-developers-to-consider-22f8d38bdfa9

WebAssembly. *Caniuse* [online]. [cit. 2021-02-10]. Dostupné z: https://caniuse.com/wasm

Přílohy

Součástí elektronické verze práce je projekt vytvořené aplikace v Blazoru v souboru zip.