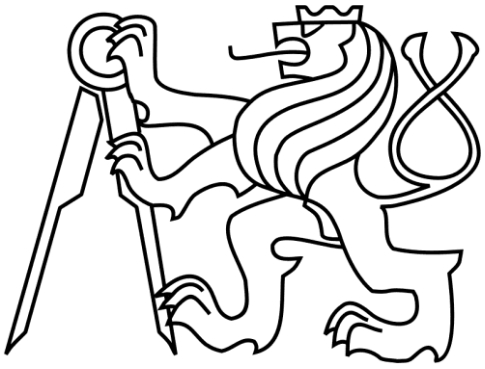
ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
FAKULTA ELEKTROTECHNICKÁ



Diplomová práce

Metodika vývoje webových mapových aplikací

Bc. Ondřej Suchý

Vedoucí práce: Ing. Radek Mařík CSc.

Studijní program: Otevřená informatika (magisterský)
Obor: Softwarové inženýrství

25.5.2017

České vysoké učení technické v Praze
Fakulta elektrotechnická

Katedra počítačů

ZADÁNÍ DIPLOMOVÉ PRÁCE

Student: Suchý Ondřej

Studijní program: Otevřená informatika
Obor: Softwarové inženýrství

Název tématu: Metodika vývoje webových mapových aplikací

Pokyny pro vypracování:

1) Seznamte se s prostředky pro vývoj interaktivních webových mapových aplikací. Prozkoumejte a porovnejte alternativy k jazyku JavaScript, posuďte vhodnost použití pro daný typ úloh.
2) Porovnejte možnosti automatizace buildu JavaScript aplikací.
3) Srovnejte knihovny pro zobrazení map v prohlížeči.
4) Seznamte se s prostředky pro testování JavaScript aplikací. Posuďte je s ohledem na tvorbu, exekuci a vyhodnocení testů.
5) Vytvořte postup pro sestavení vhodného vývojového prostředí a metodiku vývoje daného typu webových aplikací.
6) Vývojové prostředí a metodiku ověřte na reálném projektu.

Seznam odborné literatury:

[1] Johansen, Christian. Test-driven JavaScript development. Addison-Wesley Professional, 2010. ISBN: 978-0-321-68391-5
[2] Gupta, Ravi Kumar, Hetal Prajapati, and Harmeet Singh. Test-Driven JavaScript Development. Packt Publishing Ltd, 2015. ISBN: 978-1-78217-492-9
[3] Alpaev, Gennadiy. TestComplete Cookbook. Packt Publishing Ltd, 2013. ISBN: 978-1-84969-358-5

Vedoucí: Ing. Radek Mařík, CSc.

Platnost zadání do konce letního semestru 2017/2018

L.S.

prof. Dr. Michal Pěchouček, MSc.
vedoucí katedry

prof. Ing. Pavel Ripka, CSc.
děkan

V Praze dne 3.1.2017

Poděkování

V první řadě bych chtěl poděkovat Ing. Radku Maříkovi CSc. za vedení mé diplomové práce a celého projektu zaměřeného na metodiku vývoje webových mapových aplikací, za cenné rady a konstruktivní kritiku. Dále děkuji Ing. Tomáši Vlčkovi CSc. za zapůjčení potřebného softwaru a za možnost ověřit si poznatky zjištěné v této práci na reálném projektu a v neposlední řadě bych rád poděkoval své rodině a kolegům za poskytnutou podporu a motivaci.

Abstract

This thesis is focused on the methodology of web map application development. In particular, the thesis is aimed at identification of current trends in interactive web application development. Further on, the thesis analyses possible alternatives to JavaScript, i.e. the languages that are transcompiled to JavaScript, like TypeScript, Dart or CoffeeScript. Next chapters deal with possibilities of build automatization in JavaScript environment, i.e. running of build tasks, automated tests, and minification of production code. Next chapters investigate up-to-date libraries for displaying maps and geographical data in web browsers, like OpenLayers, Leaflets, etc. Eventually, the thesis explore existing options for execution of automated tests in JavaScript environment.

Key words:

Web applications, JavaScript, JavaScript alternatives, JavaScript libraries for displaying map in web browsers, build automatization in JavaScript environment, testing of JavaScript applications

Abstrakt

Tato práce se zaměřuje na metodiku vývoje a testování webových aplikací pro práci s mapami. Jejím cílem bylo zjistit, jaké jsou současné trendy pro vývoj interaktivních webových aplikací, porovnat různé alternativy jazyka JavaScript, tj. programovacích jazyků, které se kompilují do jazyka JavaScript, jako například TypeScript, Dart nebo CoffeeScript. V dalších částech se práce zaměřuje na možnost automatizace vývojového procesu JavaScriptových aplikací, tj. spuštění testů, minifikace produkčního kódu. Následuje porovnání různých knihoven pro zobrazení map ve webovém prohlížeči, jako jsou např. OpenLayers a Leaflets. Na závěr se práce věnuje tvorbě a provádění automatizovaných testů aplikací napsaných v jazyce JavaScript.

Klíčová slova:

Webové aplikace, JavaScript, alternativy JavaScriptu, JavaScriptové knihovny pro zobrazení map ve webovém prohlížeči, automatizace vývojového procesu JavaScriptových aplikací, testování JavaScriptových aplikací

Prohlášení

Prohlašuji, že jsem práci vypracoval samostatně a že jsem uvedl veškeré informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze dne 25. 5. 2017 .....

Ondřej Suchý

Obsah

Seznam tabulek

Seznam obrázků

Seznam příkladů

Kapitola 1

Úvod

Webové aplikace se v poslední době stávají jedním z hlavních způsobů distribuce aplikací. Díky prudkému rozvoji internetu a webových prohlížečů, je dnes možné v prohlížeči používat software, který byl před několika lety dostupný pouze jako nativní aplikace. V dnešní době existují velmi komplexní webové aplikace jako například online kancelářské balíčky (např. Google Docs nebo MS Office), přehrávače videa či mapové aplikace, které dříve v prohlížeči nefungovaly, nebo byly velmi závislé na platformách jako je například Adobe Flash. Dnešní webové aplikace jsou postaveny na rodině technologií HTML 5 a JavaScriptu.

Pro programátora zvyklého na vývoj v objektově orientovaných jazycích typu Java nebo C#, však nemusí být používání jazyka JavaScript, který se na webu prosadil, vůbec příjemná záležitost. Jazyk JavaScript měl být jednoduchý skriptovací jazyk a vznikal původně jako prostředek pro oživení webových stránek pomocí krátkých skriptů.

Některé jeho vlastnosti mohou pomoci při debuggování, například neexistence privátních položek v objektech (vše je public), a snadná interaktivní konstrukce objektu přímo v konzoli prohlížeče. Programátor má možnost jednoduše měnit hodnoty položek objektů, může vytvářet objekty nové a snadno je podsouvat běžící aplikaci. Nepotřebuje k tomu žádné speciální nástroje, stačí ladící nástroje ve webovém prohlížeči.

Při práci na velkých projektech se programátor často zbytečně zdržuje hledáním chyb, které by v jazyku, který provádí typovou kontrolu, byly odhaleny během kompilace. Ve větších projektech může dále vadit chybějící podpora modulů, která není v jazyce JavaScript tak dobře zavedena jako třeba v ekosystému programovacího jazyka Java. Řešení existují, ale jde o dodatečné knihovny a často obtížně čitelná řešení využívají lexikálních uzávěrů (closures).

Cílem této práce je seznámit se s prostředky pro vývoj interaktivních webových aplikací. Zejména s těmi, které pracují s geografickou informací a zpracováním map. Práce porovnává vlastnosti těchto nástrojů a na základě tohoto porovnání vybírá nástroje, které tvoří soustavu co nejvíce vzájemně spolupracujících modulů vývojového prostředí, umožňující co nejvyšší stupeň automatizace vývojového procesu. Hlavním výsledkem práce pak je metodika tvorby interaktivních, zejména mapových, webových aplikací, která má umožnit vznikajícímu týmu programátorů sestavit vhodné vývojové prostředí a používat vhodné praktiky pro vývoj zmíněné třídy webových aplikací.

Mapové aplikace byly ve webových prostředích dříve řešeny pomocí technologií jako je Java Applets nebo Flash. Zobrazení pokročilých GISových dat ve webovém prohlížeči bez použití

Java Appletů, Flash a podobných technologií komplikoval nedostatečný výkon tehdejších interpreterů jazyka JavaScript a roztříštěnost podporovaných rysů jazyka JavaScript různými webovými prohlížeči. Poměrně velká komunita uživatelů mapových úloh využívajících technologie Java Aplet se v současnosti dostává do akutního problému, protože téměř všichni výrobci webových prohlížečů přestávají podporovat tuto technologii, zejména z důvodů jejích principiálně nedostatečných bezpečnostních vlastností.

Prohlížeč Google Chrome nepodporuje Java Applety od verze 45 vydané v září roku 2015[72][74]. Firefox tuto podporu vypíná na přelomu let 2016/2017[73] ve verzi 52. V případě prohlížeče Firefox může nouzovým řešením být používání ESR (Extended Support Release) verze, která bude Java Applets podporovat ještě další rok. Microsoft Edge nemá podporu Java Appletů od samotného počátku.

Uvedené aplikace proto musejí být nahrazeny aplikacemi založenými na jiné technologií. Touto technologií bude v převážné většině případů platforma HTML 5 a jazyka JavaScript. Lze tedy očekávat masivní nárůst počtu vyvíjených interaktivních mapových aplikací na platformě HTML 5 a vznik nových vývojové týmů, jejichž členové často na počátku nebudou mít potřebné zkušenosti a budou se v poměrně rozmanitém světě JavaScriptových technologií obtížně orientovat. Tato diplomová práci má ambici poskytnout takovým začínajícím týmům vodítko při sestavování vývojového prostředí a metodiky vývoje.

Předložená práce je cíleně zaměřena na cílovou skupinu programátorů, kteří jsou zvyklí programovat v prostředích objektově orientovaných, silně typových jazyků jako je Java nebo C#, tj. v jazycích, které jsou typické pro vývoj back-end částí webových aplikací. Pozornost je věnována popisu specifických vlastností platformy JavaScript a prozkoumání různých možností vývoje interaktivních webových aplikací se zaměřením na mapové aplikace.

Práce se rovněž zbývá existujícími alternativami k jazyku JavaScript a v neposlední řadě pomocnými nástroji, které lze použít pro vývoj aplikací pro platformu HTML5. Značná pozornost je pak věnována možnostem testováni interaktivních webových aplikací.

1.1 Mapové aplikace pro webové prohlížeče

Důraz je v této práci kladen na vývoj mapových aplikací a na dostupné knihovny pro práci s mapami v jazyce JavaScript. Mnoho poznatků shrnutých v této práci se však se vztahuje na i na jiné, zejména rozsáhle, projekty vyvíjené na platformě JavaScript.Dnes se i pokročilé GISové aplikace vyvíjejí v jazyce JavaScript nebo jazycích, které se do něj kompilují. Možnost implementovat tyto poměrně komplikované aplikace v jazyce JavaScript byla umožněna výrazným nárůstem výkonu moderních interpreterů JavaScript a novými technologiemi v HTML 5 jako je Canvas.

V této práci se budu zabývat právě prostředky k vývoji webových aplikací pro práci s mapami. Zejména je potřeba vyřešit otázku volby jazyka, tj. zda zvolit JavaScript nebo nějaký jazyk, který se do něj kompiluje. Dále je potřeba vyřešit volbu knihovny pro zobrazení mapových dat, jaké prostředky zvolit pro automatizaci sestavování (buildu) projektu a testování výsledné aplikace.

Tyto otázky je třeba řešit s ohledem na charakter předpokládané aplikace. Zaměřme se proto nejprve na vymezení typu vyvíjené aplikace.

1.2 Vymezení typu webové aplikace

Předpokládaným typem vyvíjené aplikace je mapová aplikace, která slouží k zobrazení mapových podkladů a dalších mapových dat. Jako předloha slouží konkrétní aplikace pro zobrazení a editaci trajektorií letových plánů pro rekreační létání, s jejímž vývojem mám vlastní praktické zkušenosti.

Daná aplikace by neměla být závislá na připojení k internetu a měla by být schopná fungovat i v intranetu, který je od internetu oddělený. Nemůžeme tedy spoléhat na využití internetových zdrojů mapových podkladů ani internetových zdrojů dalších geografických dat.

1.3 Terminologická poznámka

Při psaní textu předložené diplomové práce jsem vyšel z předpokladu, že typickým čtenářem této práce je softwarový vývojář nebo manažer vývoje software. Tvorba odborné české terminologie významně zaostává za překotným vývojem IT technologií. Pro českou odbornou informatickou komunitu je typický odborný slang prosycený anglicismy. Velice často se setkáváme s tím, že používání neustálených českých překladů některých odborných anglických termínů je překážkou vzájemného pochopení členů české informatické komunity. V předložené práci se proto nesnažím o překládání anglických termínů do češtiny za každou cenu. Tam, kde výstižnosti textu prospěje běžně programátory používaný výraz převzatý z angličtiny a není ustálený příslušný český výraz, je v této práci používán anglicismus. Příkladem může být slovo *deprecated* namísto nepoužívaného českého ekvivalentu *zavržený*, *canvas* místo *plátno* nebo *closures* místo *uzávěry*.

Pokud se v textu mluví o jazyku *JavaScript* a není-li explicitně uvedena jeho verze, je míněna verze *EcmaScript 5*, tj. verze aktuálně podporovaná širokým spektrem webových prohlížečů.

Kapitola 2

HTML 5 a webové aplikace

Webové aplikace jsou dnes velmi důležitým typem aplikací, jejichž důležitost nejspíše ještě poroste se vzestupem cloudových technologií a s neustále více a více se prosazujícím obchodním modelem 'software jako služba'. Jazyk HTML 5 přinesl jazykovou podporu pro tvorbu těchto aplikací, které jsou načítány jako webové stránky, ale v mnoha ohledech se chovají jako klasické desktopové aplikace.

2.1 Webové aplikace

Nejdříve je dobré si ujasnit, co je webová aplikace. Webová aplikace je většinou klient-server aplikace, kde klient běží v internetovém prohlížeči. Hranice mezi dynamickou webovou stránkou a webovou aplikací není ostrá. Single-page aplikace lépe spadají pod označení aplikace, protože přejímají chování desktopových a mobilních aplikací a ustupuje u nich model přecházení mezi jednotlivými webovými stránkami, který je typický pro práci s klasickými weby.

2.2 Technologie webových aplikací

Webové aplikace jsou na klientské straně založeny na 3 základních technologiích:

- Značkovací jazyk HTML (HyperText Markup Language), který popisuje sémantickou strukturu jazyka HTML (může obsahovat i tagy, které popisují vzhled dokumentu např. `<b>`, `<i>`, ty jsou ovšem deprecated)
- Kaskádové styly (CSS), které popisují způsob zobrazení elementů, zapsaných značkovacím jazykem HTML.
- JavaScript, který je programovacím jazykem webu.

Webová aplikace by se tedy dala přirovnat k dynamické webové stránce, kde HTML kóduje zobrazený obsah, kaskádové styly obsahu dodávají formu a JavaScript poskytuje dynamické chování a zajišťuje komunikaci se serverem, například pomocí technologie AJAX.

2.2.1 Značkovací jazyk HTML

Jazyk HTML se původně zrodil v roce 1990 v CERNu, výzkumném centru jaderné a částicové fyziky ve Švýcarsku poblíž Ženevy, jako nástroj pro sdílení informací mezi vědci a univerzitami po celém světě[5][6]. Textový dokument ve formátu HTML se kromě samotného textu skládá ze značek (tagů) a atributů. Tagy typicky určují, které části dokumentu patří do které sekce, odstavce nebo

odkazují na jiná místa v dokumentu, či jiné dokumenty. HTML obsahuje samotný obsah a strukturu webového dokumentu, případně aplikace. Vzhled dokumentu by neměl být definován prostředky HTML. To mají za úkol tzv. kaskádové styly. (CSS).

2.2.2 Kaskádové styly (CSS)

CSS je programovací jazyk, který se na webových stránkách a ve webových aplikacích používá pro popis vzhledu aplikace. Kaskádové styly byly vytvořeny s cílem oddělit obsah od vzhledu dokumentu. Jsou-li forma a obsah dokumentu odděleny, je možné poměrně snadno změnit vzhled prezentovaného dokumentu tzv. stylováním dokumentu, tedy modifikací příslušného kaskádového stylu. Stylování se běžně používá např. pro přizpůsobení vzhledu dokumentu zobrazovacím schopnostem používaného prezentačního zařízení. Jiný vzhled bude mít dokument na na zařízeních s velkým rozměrem displeje (tj. na klasickém počítači či tabletu), jiný na mobilu, nebo v tištěné podobě.

2.2.3 JavaScript

JavaScript je programovacím jazykem webu. Jedná se o vysokoúrovňový, dynamický, netypovaný, interpretovaný jazyk, který je vhodný k objektovému i funkcionálnímu programování.

Jeho jméno „JavaScript“, je zavádějící. JavaScript nemá s Javou, kromě z části podobné syntaxe, nic společného. Syntaxe obou jazyků, jak Javy, tak JavaScriptu, je inspirována jazykem C. Jméno bylo zřejmě zvoleno ve snaze svést se na vlně popularity jazyka Java. JavaScript byl původně vyvinut pod jménem Mocha, poté oficiálně pojmenován LiveScript a až poté přišlo jméno JavaScript[9]. Standardizovaná verze jazyka JavaScript se jmenuje ECMAScript, podle evropské asociace ECMA (European Computer Manufacturers Association) a společnost Microsoft si svoji verzi pojmenovala JScript, aby se vyhnula možnému soudnímu sporu se společností Sun Microsystems, která měla ochrannou známku na JavaScript[10].

JavaScript je jeden z nejrozšířenějších programovacích jazyků, který dominuje skriptování na webových stránkách, používá se na desktopech[1] i serverech[2]. Téměř každý počítač, chytrý mobilní telefon, tablet nebo hrací konzole má v sobě nainstalovaný alespoň jeden interpret nějaké variace jazyka JavaScript.

Na chytrých telefonech nebo v počítačích není JavaScript používán pouze v prohlížečích, ale je ho možné použít i k vývoji hybridních aplikací, tj. aplikací, které nejsou ani zcela webové ani zcela nativní. Takové aplikace jsou postaveny na webových technologiích HTML, CSS a JavaScript, layout je renderován pomocí webových komponent (WebView), avšak nejde o klasické webové aplikace, protože se distribuují jako nativní aplikace a mají přístup k nativním aplikačním programovacím rozhraním (API) daného zařízení. Příkladem tohoto přístupu může být framework Apache Cordova [17] [18] na mobilních telefonech a Electron[1] framework na stolních počítačích.

Přestože nepochybně je JavaScript jeden z nejrozšířenějších jazyků na webu, je mnohými zatracován jako nepochopený programovací jazyk[15], ať už kvůli různým pastem na programátory, nižší rychlosti vykonání nebo jeho dynamické povaze. Podle mnohých jde o interpretovaný jazyk určený pro drobné skripty a ne pro obrovské aplikace, které mají statisíce nebo i několik milionů

[1] Electron framework - https://electron.atom.io/

[2] Node.js – běhové prostředí pro JavaScript - https://nodejs.org/en/

řádků. JavaScript nepodporuje výpočet ve více vláknech, nebo alespoň ne v té formě v jaké se používá v jazycích jako je Java či C#. V posledních verzích prohlížečů se objevuje podpora práce ve více vláknech ve formě WebWorkers API[16], ale komunikace s hlavním vláknem probíhá pouze pomocí posílání zpráv a výkonnostní penalizace za používání této technologie je poměrně velká ve srovnání se sdílením paměti mezi vlákny.

Jazyk JavaScript se stále velmi rychle vyvíjí. S rychlým vývojem ovšem přichází i velká roztříštěnost mezi prohlížeči. Většina dnešních prohlížečů (2017) podporuje alespoň ECMAScript v páté edici ES5[19], ovšem existuje mnoho uživatelů, kteří používají starší prohlížeče. Některé problémy lze řešit pomocí tzv. „polyfills". Jde o kód v jazyku JavaScript, který zjistí, zda prohlížeč danou funkci podporuje a pokud ne, tak se danou funkci snaží doplnit pomocí technologií, které jsou v daném prohlížeči dostupné. Bohužel ne vždy to jde a někdy je výkonnostní penalizace takového postupu poměrně velká.

V následujícím textu se budu zaměřovat hlavně na JavaScript, který je použitý v současných prohlížečích (tj. ES5, který má širokou podporu na počátku roku 2017), ovšem budu zmiňovat i některé vlastnosti, které jsou dostupné v novějších verzích JavaScriptu.

2.2.3.1 JavaScript – deklarace proměnných

Deklarace proměnných v jazyce JavaScript je možná třemi způsoby, pomocí klíčového slova **`var`**, bez klíčového slova, to ovšem není doporučováno, a klíčovým slovem **`let`**.

- Nejpoužívanější a doporučovaný způsob v ES5 je deklarace pomocí klíčového slova **`var`**. Tuto syntaxi lze použít pro deklaraci jak globálních, tak lokálních proměnných. Tento způsob je v prohlížečích široce podporován. Pokud je proměnná deklarovaná pomocí klíčového slova „**`var`**" mimo jakoukoli funkci jde o globální proměnnou, která je dostupná, kdekoli v kódu v aktuálním dokumentu. Pokud je proměnná deklarovaná klíčovým slovem **`var`** v nějaké funkci, pak je proměnná dostupná pouze v té funkci, jde o lokální proměnnou. U proměnných v JavaScriptu, které byly deklarovány slovem **`var`**, nemá blok kódu ohraničený složenými závorkami vliv na platnost proměnné.
```
var x = 10;
var x2; //proměnou lze deklarovat bez inicializace
```

- Nedoporučený způsob deklarace globální proměnné je pouze pomocí deklarace hodnoty. Tento způsob vyvolá při použití tzv. `strict` modu varování. Programátorovi se může velmi snadno stát, že zapomene napsat klíčové slovo var a omylem deklaruje globální proměnou, proto tento způsob není doporučován a mnohými nástroji je považován za velmi podezřelý, pokud je v kódu použitý.
```
y = 10;
```

- Posledním způsobem je deklarace pomocí klíčového slova `let`, jde o způsob, který zatím není v produkčním kódu, který je distribuován do prohlížečů, příliš rozšířen. Jeho podpora byla přidána do specifikace jazyka ECMAScript 2015/ES 6 a tato poměrně nová verze JavaScriptu ještě není podporovaná všemi prohlížeči. Deklarace pomocí klíčového slova **`let`** přináší do nejnovějších verzí JavaScriptu proměnné, jejichž platnost se řídí podle bloku kódu, ohraničeného pomocí složených závorek a ne podle funkce. Platnost proměnné podle bloku je vlastnost, na kterou jsou programátoři, pracující s jazyky vycházející z rodiny C, zvyklí.
```
let z = 10;
```

Pokud proměnné nejsou inicializovány, mají hodnotu „`undefined`“. Ovšem zvláštností JavaScriptu je tvz. „function hoisting“. Tj. proměnné deklarované pomocí klíčového slova **`var`**, je možné používat od začátku funkce, i když proměnná ještě nebyla deklarována. Takováto proměnná má před deklarací vždy hodnotu `undefined`, přestože může být později inicializována. U proměnných deklarovaných pomocí klíčového slova **`let`**, není možné používat proměnné před jejich deklarací.

2.2.3.2 JavaScript – datové typy

JavaScript má pět nebo šest primitivních datových typů:

- **Boolean**, který může nabývat hodnot `true` a `false`.
- **null** – prázdná hodnota
- **undefined** – nedefinovaná hodnota
- **Number** – číselný datový typ, který se používá jak pro celá čísla, tak pro desetinná čísla.
- **String** – textové řetězce
- V ECMAScript 2015, přibyl nový typ **Symbol**, který se používá pro unikátní instance, které jsou neměnné (immutable)

Další datové typy v JavaScriptu jsou složené, mohou v sobě obsahovat primitiva, další objekty, pole nebo funkce.

- **Object** – Objekty jsou asociativními poli, mohou obsahovat atributy a funkce, obojí je možné přidávat dynamicky. Objekty v JavaScriptu podporují prototypovou dědičnost. Koncept tříd byl přidán až do ES6 (ECMAScript 2015).

  Příklad 2.1 – Objekt v JavaScriptu

  ```
  var myObject = {
      strProp: "myString",
      numProp: 42,
      funcProp: function(){
          alert("myObject.funcProp() byl zavolan.");
      }
  }
  ```

- **Array** – Pole je v JavaScriptu vysokoúrovňový objekt ve stylu listu.
  ```
  var colors = ['red', 'green', 'blue'];
  ```

- **RegExp** – Regulární výrazy jsou také objekty, slouží k vyhledávání vzorů v řetězcích.
  ```
  var regE = /a+hoj/;
  ```

Funkce hrají v jazyku JavaScript velmi důležitou roli. Kromě volání je možné s funkcemi pracovat jako s objekty, ukládat je do proměnných, použít je jako parametr jiné funkce a lze na proměnných typu funkce volat funkce, které jsou pro typ funkce definovány, jako například funkce `call()` a `bind()`.

JavaScript je dynamicky typovaný jazyk, takže datový typ proměnné se nedeklaruje. Datové typy jsou automaticky konvertovány, když je to potřeba. Toto chování ovšem může programátora někdy zmást více, než nutnost konverze v jiných jazycích.

Funkce lze deklarovat několika způsoby. Můžeme je definovat napřímo, jak je vidět na příkladu funkce `absolutníHodnota()`. Funkce je pak přidána do aktuálního scope, to může být buď globální scope, pokud nejsme v žádné funkci, nebo lokální scope aktuální funkce.

Další možností je deklarovat funkci a uložit je jako hodnotu proměnné nebo jako atribut objektu, jak je vidět na příkladu funkce `factorial()`. Jméno uvedené za klíčovým slovem `function`, v tomto příkladu `fact()`, platí pouze uvnitř právě definované funkce v případě definice funkce tímto způsobem.

Příklad 2.2 – Funkce v JavaScriptu

```
function abolutniHodnota(cislo){
    if(cislo<0){
      return -cislo;
    }
    return cislo;
}

var faktorial = function fact(n){
    if(n<2){
        return 1;
    }
    return n * fact(n-1);
}

var objekt = {};
objekt.factorial = faktorial;
```

V jazyku JavaScript existují dvě sady operátorů pro porovnávání proměnných. Operátory „!=„ a „==„ porovnávají hodnotu, ovšem pokud se jedná o hodnoty dvou různých typů, tak se snaží hodnotu automaticky převést. Na rozdíl od operátorů „!== “ a „===“, které vyžadují, aby byla stejná jak hodnota, tak její typ.

2.2.3.3 JavaScript – Closures

Velmi důležitým konceptem v jazyku JavaScript jsou tvz. closures, v češtině někdy označované jako uzávěry. Vzhledem k tomu, že JavaScript před ES6 nepodporuje moduly, využívají se právě closures ke členění kódu do jakýchsi modulů.

Pokud bychom potřebovali v naší webové aplikaci počítadlo, které by počítalo zavolání nějaké funkce. Nejjednodušší je si tento stav pamatovat v globální proměnné.

Příklad 2.3 – Počítadlo s globální proměnnou

```
var pocitadlo = 0;

function pridejJedna() {
    pocitadlo += 1;
}

pridejJedna();
pridejJedna();
```

```
pridejJedna();

// pocitadlo ma ted hodnotu 3
```

Ovšem v případě větších aplikací je mnohem lepší zvolit nějaký modulární přístup, který by zapouzdřil jednotlivé funkční celky. Pro použití closures v JavaScriptu je velmi důležité, že pokud v JavaScriptu deklarujeme nějakou funkci uvnitř jiné funkce, tak vnitřní funkce bude vytvořená při každém volání funkce vnější. Velmi zajímavou vlastností JavaScriptu je, že proměnné vnější funkce existují nejen po dobu volání funkce vnější, ale i po dobu existence funkce vnitřní. Dokonce i po ukončení konkrétního volání vnitřní funkce, je stav proměnných zachován a lze ho využít pro další volání vnitřní funkce. Stav proměnných vnější funkce, který náleží ke konkrétní instanci vnitřní funkce, zůstává zachován, dokud lze volat danou instanci vnitřní funkce.

Příklad 2.4 – Počítadlo s využitím closures

```
var pridejJednaAVratVysledek = (function () {
    var pocitadlo = 0;
    return function () {return pocitadlo += 1;}
})();

pridejJednaAVratVysledek();
pridejJednaAVratVysledek();
pridejJednaAVratVysledek();

// pocitadlo ma ted hodnotu 3
```

2.2.3.4 JavaScript – Proměnná this a funkce bind

Hodnota **this** je v jazyku JavaScript nastavená na objekt, který danou funkci volá. Tj. pokud je funkce v globálním scope, bude v prohlížeči **this** většinou nastaveno na objekt window. Pokud používáme `strict` mode, tak `this` bude nastaveno na `undefined` u funkcí definovaných globálně a u anonymních funkcí, které nejsou vázané na žádný objekt.

Velmi důležité je, že hodnota **this** je nastavena až při volání funkce. Tj. pokud je daná funkce volána níže uvedeným způsobem, kde „`object`“ je objekt, který bude použitý jako hodnota **this**, nenastane žádný problém.

```
object.myFunction();
```

Ovšem problém s hodnotou **this** může nastat, například pokud použijeme funkci jako callback, pokud je funkce použitá uvnitř closure, pokud je funkce přiřazena proměnné a pak zavolána z této proměnné, případně pokud si jeden objekt půjčuje metodu jiného objetu[14].

Řešením tohoto problému může být funkce `bind()`, která se volá na původní funkci a vrátí novou funkci, která má jako hodnotu **this** napevno přiřazený objekt, který je parametrem funkce `bind()`.

Příklad 2.5 – Použití funkce bind

```
var modul = {
  jmeno: "MojeJmeno",
  getJmeno: function() { return this.jmeno; }
```

```
};

modul.getJmeno(); // vrátí MojeJmeno

var ziskejJmeno = modul.getJmeno;
ziskejJmeno();
// vrátí undefined, protože funkce ziskejJmeno
// je zavolána s globálním scope

var ziskejJmenoBind = ziskejJmeno.bind(modul);
ziskejJmenoBind(); // vrátí MojeJmeno,
//protože u funkce ziskejJmenoBind byl napevno
// nastaven this na objekt modul.
```

2.3 HTML 5

HTML s sebou ve své páté verzi přinesl spoustu novinek, jako jsou například tagy <video>, pro zobrazení videa, <audio>, pro zvuk a <canvas>, pro renderování vektorové grafiky a bitmapových obrázků. Tyto nové tagy umožňují nahradit technologie jako je Adobe Flash, Microsoft Silverlight nebo Java applety, které byly dříve jedinou možností, jak zobrazit dynamický a multimediální obsah na webové stránce nebo ve webové aplikaci. Na rozdíl od předchozích technologií, které pracovali jako plug-in, je podpora pro HLTML 5 přímo zabudovaná v prohlížečích. To sebou přináší poměrně velký problém s roztříštěností podpory těchto nových vlastností ve webových prohlížečích.

Na druhou stranu aplikace postavené nad skupinou technologií známou jako HTML 5 nejsou závislé na konkrétní platformě, dodavateli, ani na jedné implementaci. To byl problém předešlých technologií, které byly více či méně závislé na jedné implementaci od jedné firmy. Flash byl závislý na společnosti Adobe, Silverlight na společnosti Microsoft a Java applety na společnosti Sun, později Oracle.

Nezávislost na jednom dodavateli je garantovaná i tak, že HTML 5 specifikace není považována za hotovou, dokud neexistují alespoň dvě nezávislé a kompletní implementace [4]. To má zajistit, aby bylo možné finální specifikaci dobře implementovat v prohlížecích a používat ji při návrhu webových stránek a aplikací. Webové prohlížeče jsou dostupné na široké škále zařízení od stolních počítačů, přes tablety až po mobilní telefony, tudíž vývojář webové aplikace může jednou aplikací pokrýt širokou škálu zařízení.

2.4 Řešení rozdílné úrovně podpory webových technologií v prohlížečích

HTML 5 je tzv. living standard, který se neustále vyvíjí a přidávají se do něj nové možnosti. Webové aplikace musí počítat s tím, že nové funkce nemusí být v daném prohlížeči k dispozici. Nabízejí se dva přístupy k řešení tohoto problému, které se v praxi dost často kombinují v závislosti na situaci[20]:

- **Graceful degradation** – postupná degradace, tj. aplikace by měla poskytovat nějakou úroveň funkčnosti v nových prohlížečích, ale v případě, že prohlížeč moderní vlastnosti nepodporuje, tak se spustí jednoduší verze, která by měla umožňovat alespoň základní funkcionalitu bez pokročilejších funkcí, pokud je to možné. Tudíž jako výchozí stav je brán pokročilý prohlížeč, který

moderní funkce podporuje, a pak se přidávají různé opravy, aby systém byl použitelný i ve starších prohlížečích byť třeba za cenu nižšího uživatelského komfortu nebo omezené funkcionality.

- **Progressive enhancement** – postupné vylepšení, tj. aplikace poskytuje základní funkce, které jsou postavené na základních technologiích dostupných téměř ve všech prohlížečích a pokud se zjistí, že prohlížeč podporuje nějakou moderní funkci, tak se aktivuje tato moderní funkce, ta je ovšem přidána jako doplněk k základnímu workflow.

Kapitola 3

Alternativy k jazyku JavaScript

Z mé osobní zkušenosti, z diskusí s kolegy i z různých ohlasů na internetu vím, že pokud přijdete do místnosti plné vývojářů, kteří programují hlavně v C# nebo v Javě a začnete mluvit o programování v jazyku JavaScript, nesetkáte se většinou s velkým nadšením. Vývojáři, kteří jsou zvyklí na podporu vývojářských nástrojů v Javě a C#, si poměrně obtížně zvykají na JavaScript, který je velmi dynamický, možnosti našeptávání a typové kontroly u něho nefungují tak dobře jako u staticky typovaných jazyků jako je C# a Java. Prototypová dědičnost, kterou jazyk JavaScript používá, není mezi typickými vývojáři, kteří se nespecializují na vývoj v jazyku JavaScript, velmi oblíbená.

Právě z tohoto důvodu vzniklo tolik alternativ a vylepšení jazyka JavaScript, které se snaží nástroje z prostředí, jako je Java a C#, přenést do světa jazyka JavaScript. Dokonce existuje mnoho desítek projektů, které se pokouší o kompilaci z jiného jazyka do jazyka JavaScript[21], pro tyto nástroje, které kompilují z jednoho jazyka do jiného, se používá výraz „transpiler“, „transcompiler“ případně „source-to-source compiler“. V této kapitole si představíme některé z nich a porovnáme jejich schopnosti.

3.1 TypeScript

TypeScript je programovací jazyk, který byl vyvinutý společností Microsoft. Jazyk byl uvolněn jako open-source pod Apache licencí verze 2.0, tudíž při jeho používání není vývojář odkázán pouze na nástroje od společnosti Microsoft.

TypeScript je nadmnožinou programovacího jazyka JavaScript. Konkrétně jde o nadmnožinu moderní verze jazyka JavaScript, to jest ECMAScript 6 (ES6). Oproti nejvíce rozšířené verzi jazyka JavaScript ES5 přidává možnost statického typování a objektového programování se třídami, tak jak jej známe z jazyků jako je Java či C#. Pokud provádíme vývoj, tak můžeme výsledný kód zatím kompilovat do ES5 a v budoucnosti přejít na ES6 jednoduchou změnou parametru TypeScript kompilátoru. Díky tomu, že jde o nadmnožinu jazyka JavaScript, jsou až na výjimky existující programy v jazyku JavaScript platnými programy jazyka TypeScript. To usnadňuje přechod projektů vyvíjených v jazyku JavaScript na TypeScript. Dokonce je možné do TypeScript projektů, přímo vkládat zdrojové kódy v jazyce JavaScript, jen je nutné během kompilace použít parametr `--allowJs`. Díky této kompatibilitě jazyka TypeScript s jazykem JavaScript je možné již probíhající projekt převést poměrně hladce, případně je možné přechod provádět postupně během vývoje a postupně procenta JavaScript kódu snižovat a časem projekt přeměnit na čistý TypeScript.

Při použití knihoven v jazyce JavaScript programátor může používat takzvané definiční soubory s příponou `.d.ts`, které lze přirovnat k hlavičkovým souborům v jazycích C/C++. V definičních souborech je definované API, které daná knihovna nabízí a umožňuje využívat výhody typové kontroly v TypeScript kompilátoru (tsc) i při práci s danou knihovnou. Uživatel navíc nemusí tyto definice psát sám, ale může využít definice, které jsou již pro populární knihovny napsány, případně vygenerovány. DefinitelyTyped je depozitář, ve kterém se nacházejí tisíce definičních souborů pro knihovny různých verzí [24][25]. Použití definičních souborů zajistí typovou kontrolu při kompilaci a našeptávání ve vývojovém prostředí i pro knihovny, které jsou v jazyku JavaScript.

Velkou výhodou pro vývojáře, kteří jsou obeznámeni s jazykem JavaScript je, že proces kompilace je pro programátora velmi transparentní, kompilovaný JavaScript je velmi dobře mapovatelný na původní zdrojový kód v jazyce TypeScript, tj. pro programátora je velmi snadné říct, které řádky ve zdrojovém TypeScript kódu odpovídají kterým řádkům ve výsledném JavaScript kódu. Pokud programátor v konzoli prohlížeče zjistí, kde v JavaScript kódu se stala chyba, tak je pro něj velmi snadné ji opravit ve zdrojovém kódu psaném v jazyce TypeScript. Naproti kód produkovaný například níže zmíněným kompilátorem Dart, může být zdrojovému kódu podobný mnohem méně, což může případné krokování a zjišťování zdroje chyb významně zkomplikovat. Transparentní proces kompilace bych hodnotil jako jednu z největších výhod jazyka TypeScript oproti jeho konkurentům. I na hlavní straně jazyka TypeScript se dozvíme, že jde o „JavaScript that scales.“[92], tj. „JavaScript, který lze škálovat“. Tím tvůrci jazyka TypeScript naráží na to, že jazyk TypeScript je zamýšlen pro větší projekty, které je velmi těžké dlouhodobě vyvíjet a udržovat, pokud jsou napsány v čistém jazyce JavaScript.

TypeScript compiler (tsc) zároveň generuje mapovací soubory (`.js.map`), díky tomu je možné v prohlížečích debuggovat přímo TypeScriptové zdrojové kódy, pokud jsou na serveru dostupné a pokud použitý prohlížeč použití mapovacích souborů podporuje. Možnost použití mapovacích souborů je ve výchozím nastavení podporována například v prohlížeči Google Chrome. Uživateli stačí otevřít Developer Tools a je možné debuggovat přímo zdrojové kódy v jazyce TypeScript. Ovšem ani nemožnost použít při debuggování mapovací soubory (source mapping) nepředstavuje pro programátora problém, díky transparentní kompilaci jazyka TypeScript do JavaScript kódu.

3.1.1 Deklarace proměnných v jazyku TypeScript

Doporučeným způsobem deklarace proměnných je použití klíčového slova `let`, které vytvoří proměnou, která bude mít platnost v rámci bloku kódu, tj. block scoped. Pokud v rámci bloku deklarujeme vícekrát tu samou proměnou, tak se kompilátor tsc postará o to, aby proměnné byly oddělené. Z pohledu programátora bude vnější proměnná v bloku zastíněna vnitřní proměnnou, to bude ve výsledném JavaScript kódu zajištěno tím, že vnitřní proměnná bude ve vygenerovaném kódu přejmenována.

Je možné používat i klíčové slovo `var`, potom ovšem bude mít proměnná platnost v rámci funkce, jak je běžné v JavaScript kódu. V tomto případě se opakované deklarace se přepisují. Proměnou deklarovanou pomocí klíčového slova `var` je také možné používat už před samotnou definicí, stejně jako v jazyku JavaScript, ovšem pro programátory z prostředí Java či C# je toto chování překvapivé a může vést ke zbytečným chybám.

Na níže uvedeném příkladu je vidět deklarace proměnné `test`, která je deklarovaná pomocí klíčového slova `let`, a `testVar`, která je deklarovaná pomocí klíčového slova `var`. Proměnná

`test` se chová podobně jako například proměnné v jazyce Java, tj. vnější proměnná je v bloku zakryta, ale její hodnota je po skončení bloku dále přístupná. Naproti tomu proměnná, která byla deklarována pomocí `var`, je v bloku přepsána a její původní hodnota je ztracena.

Příklad 3.1 – TypeScript: Deklarace proměnné pomocí var a let, moduly

```
module Test{
    let test = "test";
    var testVar = "testVar";

    for(let i = 0; i < 10; i++){
        let test = "test2"
        var testVar = "testVar2";
    }

    console.log("test=" + test + " testVar=" + testVar);
    //test=test testVar=testVar2
}
```

3.1.2 Moduly v jazyku TypeScript

V další ukázce kódu je vidět, co se stalo při kompilaci do JavaScriptu. Moduly jsou přepsány na JavaScriptové IIFE (Immediately-Invoked Function Expression), čímž je zaručeno zapouzdření kódu v daném modulu.

Příklad 3.2 – Zkompilovaný JavaScript: Deklarace proměnné pomocí var a let, moduly

```
var Test;
(function (Test) {
    var test = "test";
    var testVar = "testVar";
    for (var i = 0; i < 10; i++) {
        var test_1 = "test2";
        var testVar = "testVar2";
    }
    console.log("test=" + test + " testVar=" + testVar);
})(Test || (Test = {}));
```

Pokud potřebujeme, aby nějaká funkce, třída, interface, konstanta nebo proměnná byla viditelná i mimo modul uvedeme ji klíčovým slovem `export` a pak k ní můžeme přistupovat i mimo daný modul. Pro přístup k exportovaným součástem modulu používáme tečkovou notaci. Moduly je možné do sebe libovolně zanořovat, mít jeden modul deklarovaný přes několik souborů.

Příklad 3.3 – TypeScript: Export proměnné a funkce

```
module Test{
    export let exportedVariable = "testExported";
    export function exportedFunction() {
        //Function code
    }

    let internalVariable = "testInternal";
```

```
    function internalFunction() {
        //Function code
    }
}
Test.exportedFunction();
Test.exportedVariable = "okItWasExported";
```

Příklad 3.4 – Zkompilovaný JavaScript: Export proměnné a funkce

```
var Test;
(function (Test) {
    Test.exportedVariable = "testExported";
    function exportedFunction() {
        //Function code
    }
    Test.exportedFunction = exportedFunction;
    var internalVariable = "testInternal";
    function internalFunction() {
        //Function code
    }
})(Test || (Test = {}));
Test.exportedFunction();
Test.exportedVariable = "okItWasExported";
```

3.1.3 Datové typy v jazyku TypeScript

TypeScript je typovaný jazyk, ve výše uvedeném příkladu jsme ovšem žádné typy uvádět nemuseli, protože z daného kódu je možné odvodit, že proměnné budou typu `string`. Pokud bychom se někdy později pokusili do proměnné typu `string` přiřadit například typ `number`, tak při kompilaci obdržíme chybu. Pokud bychom ovšem do této proměnné potřebovali dávat řetězce i čísla, tak ji lze deklarovat takto:

```
let jmenoPromene: string|number = „Vychozi string hodnota“;
```

Typy je možné deklarovat explicitně, jak u proměnných, parametru, návratových hodnot funkcí a atributů tříd. Typ se píše za jménem proměnné a je od ní oddělen dvojtečkou. V níže uvedeném případě funkce není nutné definovat návratovou hodnotu funkce `naDruhou`, TypeScript kompilátor si výsledný typ sám odvodí z deklarace typů vstupních parametrů, ovšem pokud typ výstupu uvedeme, tak tato informace navíc slouží jako dodatečná kontrola, zda je výstup opravdu to, co se očekává.

Příklad 3.5 – Datové typy v jazyku TypeScript

```
let promena: string;

function naDruhou(zaklad: number): number {
    return zaklad * zaklad;
}
```

TypeScript používá stejné typy, které podporuje JavaScript, tj. `boolean`, `number` (stejně jako v jazyku JavaScript jsou zde pouze čísla s plovoucí řádovou čárkou) a `string`. Na kterých lze volat stejné metody jako na stejných typech v jazyku JavaScript.

Pole lze zapsat dvěma způsoby `number[]` nebo `Array<number>`. Pokud potřebujeme vytvořit pole s předem známým počtem elementu, které obsahuje více typů, můžeme použít typ **tuple** `[string, number]`, tento příklad datového typu **tuple** označuje pole o dvou prvcích, kde první je řetězec a druhý je číslo.

TypeScript navíc podporuje typ **`enum`**, což je výčtový typ, který je interně reprezentován čísly, ovšem je možné za běhu snadno získávat i řetězcové reprezentace jednotlivých čísel. Jak je vidět v následujícím případu při kompilaci vznikne JavaScript objekt, který mapuje jak čísla na řetězce, tak řetězce na čísla.

Příklad 3.6 – Enum v jazyku TypeScript

```
enum Barva {Cervena = 1, Modra, Zelena, Zluta};
let bar: Barva = Barva.Zelena;

//Zkompilovaný JavaScript
var Barva;
(function (Barva) {
    Barva[Barva["Cervena"] = 1] = "Cervena";
    Barva[Barva["Modra"] = 2] = "Modra";
    Barva[Barva["Zelena"] = 3] = "Zelena";
})(Barva || (Barva = {}));
var bar = Barva.Zelena;
```

Další typ, který má TypeScript k dispozici je typ `any`, který mají všechny proměnné, u kterých není uvedený typ a nelze typ odvodit z kontextu. Při použití typu `any` přicházíme o výhody typové kontroly, ale při použití externího kódu napsaného v jazyku JavaScript to může být nutné. Při kompilaci je možné použít parametr `--NoImplicitAny`, který zajistí, že pokud nebude výslovně definován typ `any`, bude použití proměnné bez uvedení typu v situaci, kde není možné typ odvodit z kontextu, bráno jako chyba.

Pokud nějaká proměnná, atribut nebo prvek pole může obsahovat například číslo nebo řetězec, tak v jazyku TypeScript lze jednoduše definovat výčet typů, kterých může daná proměnná nabývat, jednotlivé typy je nutné oddělovat symbolem „|“. V tomto případě například:

```
let promenna: number|string; //proměnná, která je číslo nebo řetězec
let pole: Array<number|string>; //pole, do kterého lze dávat čísla a řetězce
```

Stejně jako JavaScript má TypeScript dva speciální typy **Null** a **Undefined**, které mají hodnoty `null` a `undefined`. Do proměnných všech typů jde dosadit hodnoty `null` a `undefined` a dříve nebylo možné tyto speciální hodnoty z ostatních typů vyjmout. Ovšem od verze TypeScript 2.0 je možné při kompilaci použít přepínač `--strictNullChecks`, který zapíná nový „strict null checking mode“, tj. mód, ve kterém tyto dvě hodnoty nejsou součástí rozsahu hodnot všech typů a pokud chceme tyto speciální hodnoty do nějaké proměnné dosazovat, musíme to výslovně specifikovat, což přináší lepší možnosti typové kontroly.

V následujícím příkladu je vidět, že proměnná `x`, může nabývat pouze číselných hodnot, na rozdíl od proměnných `y` a `z`. Pokud bude proměnná `x` nebo `y` použita před přiřazením hodnoty,

projeví se to chybou při kompilaci, protože nedefinované proměnné mají hodnotu `undefined` a ta není pro proměnné `x` a `y` přípustná.

Příklad 3.7 – TypeScript: Strict null checking mode

```
let x: number;
let y: number | null;
let z: number | undefined;
```

Pokud bude například některá funkce vracet hodnotu typu `number|null` a bude na výsledku proveden test, zda hodnota není `null`, (tj. `result !== null`) tak ve větvi po úspěšně provedeném testu kompilátor počítá s tím, že hodnota takové proměnné musí být typu `number`, když typ `null` byl vyloučen testem.

Pokud v našem projektu použijeme při kompilaci přepínač `--strictNullChecks`, tak tím zabráníme vzniku poměrně častých chyb s hodnotami `null` a `undefined`. Osobně bych řekl, že práce s odstraňováním chyb spojených s `null` a `undefined` může být výrazně větší než práce navíc při uvádění typů, proto bych použití přepínače `--strictNullChecks` u nových projektů určitě doporučil. U rozsáhlých už existujících projektů to může být přepisování a připisování typových definic poměrně náročné, ovšem pokud práce na daném projektu bude ještě delší dobu pokračovat, tak bych řekl, že se použití toho přepínače vyplatí. Zároveň to může vést i k odstranění přebytečných kontrol, protože data je potřeba kontrolovat jen jednou, když vstupují do aplikace, a tím pádem i ke zrychlení naší aplikace.

3.1.4 Třídy v jazyce TypeScript

JavaScript před ES6 podporuje pouze dědičnost založenou na prototypech, ovšem tento typ dědičnosti nemusí vyhovovat všem, proto TypeScript podporuje také třídy a interface. Třídy v TypeScriptu podporují klasickou třídní dědičnost, **atributy jsou public, pokud není uvedeno jinak**. Pokud je nějaký atribut nebo metoda označena jako `private` nebo `protected`, je k nim možné přistupovat pouze v rámci třídy pro `private` atributy a metody, případně v rámci třídy a jejich potomků v případě `protected` atributů a metod. Po kompilaci do JavaScriptu (ES3 a ES5) jsou atributy a metody třídy dostupné, vzhledem k tomu, že JavaScript nemá privátní atributy objektu. Ovšem kontrola při kompilaci se postará o to, aby privátní atributy nebyly využívány v rámci kódu v jazyku TypeScript. Třídy v jazyku TypeScript podporují i abstraktní třídy, abstraktní metody a abstraktní atributy.

Třída může, ale nemusí, interface implementovat, tj. nemusí u ní být uvedeno `implements Interface`. Pro kontrolu se používá tvz. „duck typing“, tj. pokud má třída všechny atributy daného interface a ty mají správné typy, tak je přiřazení dané třídy do proměnné, která má daný interface definovaný jako svůj typ, v pořádku. Interface se používá pouze během kontroly při kompilaci a do zkompilovaného kódu se nepřikládá.

Příklad 3.8 – Třída v jazyku TypeScript

```
//Příklad interface
interface TestInterface{
    getPrivatniAtr(): string;
}
```

```
//Deklarace třídy
class Trida {
    private privatniAtr: string;
    private static statickaHodnota = "staticka hodnota";

    constructor(parametrKonstruktoru: string) {
        this.privatniAtr = parametrKonstruktoru;
    }

    public getPrivatniAtr() {
        return this.privatniAtr;
    }

    public static getStatickaHodnota() {
        return Trida.statickaHodnota;
    }

}

//Použití třídy
let instanceTridy = new Trida("Je to trida");
let privatniAtr = instanceTridy.getPrivatniAtr();
```

3.2 Dart

Dart je programovací jazyk vyvinutý společností Google. Jde o objektově orientovaný jazyk, se třídami a jednonásobnou dědičností, který má podobnou syntaxi jako jazyky Java nebo C#. Při srovnání s jazykem TypeScript, je mnohem více vzdálen od jazyka JavaScript a kompilace už není tak transparentní jako v případě jazyka TypeScript. Použití externích JavaScript knihoven je v jazyce Dart komplikovanější než v jazyce TypeScript, je nutné definovat rozhraní mezi oběma jazyky, proto případný přechod projektu nemůže být tak hladký jako v případě jazyka TypeScript. Ovšem existuje nástroj js_facade_gen[3], který umí rozhraní mezi aplikací napsanou jazykem Dart a JavaScript knihovnou vygenerovat z `.d.ts` souborů napsaných pro TypeScript[57]. Tím může i jazyk Dart těžit z poměrně široké TypeScript komunity.

Kód generovaný při kompilaci ze zdrojových Dart kódů do JavaScript kódu je hůře čitelný a není tak zřejmé, jaká část zkompilovaného kódu náleží jaké části kódu před kompilací. Na druhou stranu má kompilátor jazyka Dart větší možnosti v oblasti optimalizací. Dříve byly plány na zahrnutí Dart virtual machine (Dart VM) přímo do prohlížeče Chrome. Pak by Dart aplikace mohly být v Chrome rychlejší než v ostatních prohlížečích, kde by běžela verze překompilovaná do jazyka JavaScript, tyto plány byly ovšem zrušeny a bylo rozhodnuto, že se vývoj jazyka Dart soustředí na kompilaci do jazyka JavaScript, který je podporovaný napříč prohlížeči.[58]

[3] Js_facade_gen: https://github.com/dart-lang/js_facade_gen

Všechny programy v jazyku Dart se začínají vykonávat funkcí `main()`. Neinicializované proměnné všech typů mají hodnotu `null`, což je rozdíl oproti jazykům JavaScript a TypeScript, kde je hodnota neinicializovaných proměnných `undefined`.

Vestavěné datové typy v jazyku Dart jsou:

- **`num`** – typ pro čísla, který má dva subtypy:
  - **`int`** - pro celá čísla, rozsah -2^{53} až 2^{53}
  - **`double`** - pro čísla s pohyblivou řádovou čárkou podle IEEE 754
- **`String`** - pro řetězce složené z UTF-16 znaků, lze použít jednoduché i dvojité uvozovky
- **`bool`** - pro **`true`**/**`false`** hodnoty
- **list** - pole/seznamy ohraničené hranatými závorkami
- **map** - objekty s páry klíč-hodnota
- **rune** - různé znaky a symboly v UTF-32, zapisuje se do řetězců pomocí `\uXXXX`, kde XXXX jsou 4 hexadecimální číslice, pokud je potřeba zapsat více nebo méně hexadecimálních číslic používá se syntaxe se složenou závorkou, například `\u{1f600}` je smějící se smajlík
- **symbol** – symbol objekt reprezentuje operátor nebo identifikátor, tento typ objektu není při minifikaci přejmenován

Pokud je kód překompilován do jazyka JavaScript, tak se s oběma číselnými typy pracuje jako s čísly s pohyblivou řádovou čárkou, ovšem Dart VM celá čísla podporuje, tudíž se může chování a rychlost vykonávání lišit oproti verzi překompilované do jazyka JavaScript.

Dart podporuje třídy i abstraktní třídy. Abstraktní třídy v Dart kódu jsou označeny klíčovým slovem `abstract` před jménem třídy. Abstraktní třídy mohou mít abstraktní metody, stačí jen vynechat tělo metody. Třída, která dědí od abstraktní třídy, má za jménem `extends` a jméno třídy, ze které dědí.

Každá třída zároveň automaticky definuje interface. Tudíž ostatní třídy, i když nejsou jejími potomky, mohou implementovat metody ostatních tříd, v tomto případě se za jménem třídy píše klíčové slovo `implements`. Při použití tohoto slova kompilátor zkontroluje, zda daná třída implementuje všechny atributy a metody, které má třída, která je implementována.

Příklad 3.9 – Příklad Dart

```
abstract class PocitacInstanci{
  num getPocetInstanci();
}

abstract class AbstraktniTrida implements PocitacInstanci {
   static num pocetInstanci = 0;

  AbstraktniTrida(){
    AbstraktniTrida.pocetInstanci++;
  }

  //Definice konstruktorů, atributů a metod
  num getPocetInstanci(){
```

```
    return AbstraktniTrida.pocetInstanci;
  }

  String getJmenoTridy();//Abstraktní metoda
}

class Trida extends AbstraktniTrida{
  String jmeno;

  Trida(this.jmeno);

  String getJmenoTridy(){
    return jmeno;
  }
}

void main() {
  var trida1 = new Trida("Moje trida 1");
  print(trida1.getJmenoTridy()); //Vypíše: Moje trida 1
  print(trida1.getPocetInstanci());//Vypíše: 1
  var trida2 = new Trida("Moje trida 2");
  print(trida1.getPocetInstanci());//Vypíše: 2
}
```

3.3 CoffeeScript

CoffeeScript je také programovací jazyk, který se trans-kompiluje do JavaScript kódu. Je inspirovaný jazyky Ruby a Python. Oproti jazykům TypeScript a Dart si programátor, který je zvyklý na prostředí jako je C# a Java, si bude muset zvykat na velmi rozdílný jazyk.

Už komentáře jsou dělány naprosto jinak, než je zvykem v jazycích inspirovaných jazyky z C rodiny. Jednořádkový komentář je uvozen symbolem #, více řádkový je uvozen i ukončen řádkem, kde jsou `###`.

Příklad 3.10 – Komentáře CoffeeScript

```
# Toto je jednořádkový komentář

###
Toto je víceřádkový komentář
###
```

Bílé znaky mají v jazyku CoffeeScript na rozdíl od jazyka JavaScript význam. Místo používaní složených závorek `{ }`, se blok kódu ohraničuje pomocí zanoření, tj. pomocí tabulátorů, podobně jako v jazyku Python. Není třeba psát středníky k ukončení výrazu, stačí použít konec řádku.

Funkce se definují pomocí operátoru „->“, jak je vidět na následujícím příkladu[27]. Mohou mít předefinovanou hodnotu parametrů, která se použije, pokud bude mít parametr při volání hodnotu `null` nebo `undefined`.

Příklad 3.11 – CoffeeScript: Funkce

```
# CoffeeScript
naDruhou = (x) -> x * x
naTreti   = (x) -> naDruhou(x) * x

//JavaScript
var naTreti, naDruhou;

naDruhou = function(x) {
  return x * x;
};

naTreti = function(x) {
  return naDruhou(x) * x;
};
```

`If` se může psát v post-fixovém formátu a bez použití závorek a složených závorek. Celkově je CoffeeScript mnohem více orientovaný na slova než speciální znaky. Používá se „`is`“ místo „===“, „`not`“ pro negaci a „`and`“ místo „`&&`“. Viz následující tabulka[27].

Tabulka 3.1 – CoffeeScript vs JavaScript

CoffeeScript	JavaScript
Is	===
isnt	!==
not	!
and	&&
or	||
true, yes, on	true
false, no, off	false
@, this	this
of	in
in	(nemá ekvivalent v JS)
a ** b	Math.pow(a,b)
a // b	Math.floor(a/b)
a %% b	(a % b + b) % b

CoffeeScript podporuje třídy a klasickou třídní dědičnost. Ovšem definice třídy je ohraničená pomocí zanoření, což je z pohledu programátorů, kteří jsou zvyklí na jazyky typu Java nebo C#, nezvyklé.

Příklad 3.12 – CoffeeScript: Třídy

```
class Zvire
      constructor: @jmeno ->

      pohyb: (metry) ->
            alert @jmeno + " pohyb #{meters}m."

class Pes extends Zvire
    pohyb: ->
        alert "Beh";
        super 20

punta = new Pes "Punta"
punta.pohyb()
```

Objekty je možné v jazycích CoffeeScript a JavaScript zapisovat stejně. Případně je možné stejně jako u tříd použít zanoření místo složených závorek.

Příklad 3.13 – CoffeeScript: Objekty

```
chleba = {cena: 35, jmeno: "Sumava"}

rodina =
      otec:
            jmeno: "Pavel Novak"
            vek: 40
      matka:
            jmeno: "Marie Novakova"
            vek: 3
```

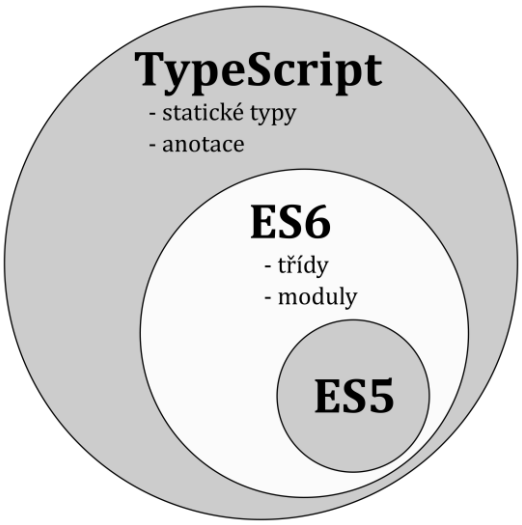
3.4 EcmaScript6

ECMAScript 6 nebo také ECMAScript 2015, je již šestou verzí standartu jazyka JavaScript. Přestože už existuje sedmá edice, šestá edice stále nemá širokou podporu mezi prohlížeči. I tak je možné některé vlastnosti z této edice používat pomocí trans-kompilátorů, kteří zkompilují novější verzi jazyka JavaScript do široce podporovaného ES5. Příkladem takového kompilátoru může být třeba Babel.

Babel umožňuje transformovat moderní JavaScript například ES6 na starší JavaScript, který je podporovaný prohlížeči. Navíc obsahuje mnoho polyfills, tj. kódu, který se snaží do staršího JavaScript kódu doplnit funkcionalitu z novější verze jazyka JavaScript.

Mezi nové vlastnosti podporované v ES6 patří“ funkce definovaná šipkou, třídy, rozšíření objektových literálů, template strings, destructuring, let a const, iterátory a mnoho dalšího.

ES6 je podmnožina toho, co nabízí TypeScript, jak je vidět na následujícím obrázku, který znázorňuje vztah mezi ECMAScript ve verzi 5, 6 a jazykem TypeScript. Tudíž pokud je náš projekt většího rozsahu nebo pokud preferujeme statické typy, tak je lepší používat TypeScript.



Obrázek 3.1 - Vztah mezi ES5, ES6 a TypeScript[59]

3.5 Google Closure

Snaží se dosáhnout stejného cíle jako kompilátor jazyka TypeScript, ovšem místo přidání typů do programovacího jazyka, jsou v Google Closure typy přidány do JSDoc dokumentace.

Toto řešení se mi nezdá vhodné, dokumentace pak může mít vliv na funkčnost kódu, protože špatně uvedený typ může při kompilaci v „ADVANCED_OPTIMALIZATIONS“ módu, který provádí velmi agresivní minifikaci kódu. Během práce v ADVANCED módu Closure kompilátor přejmenuje na co nejkratší názvy všechny proměnné, které nejsou exportovány, uvedeny jako externs nebo je Closure kompiler nepovažuje za standartní součást JavaScriptu, např. `console.log(),window.alert()` atd.

Uživatel může zvolit i mírnější formu kompilace. Nejmírnější je tzv. „WHITESPACE_ONLY“ mód, který neprovádí přejmenovávání dlouhých názvů, ale pouze odstraňuje znaky, které nemají pro vykonávání kódu žádný význam. Další mód je takzvaný „SIMPLE_OPTIMALIZATIONS“, který přejmenovává pouze lokální proměnné funkcí, které nejsou mimo tyto funkce dostupné.

3.6 Proč jsem zvolil TypeScript

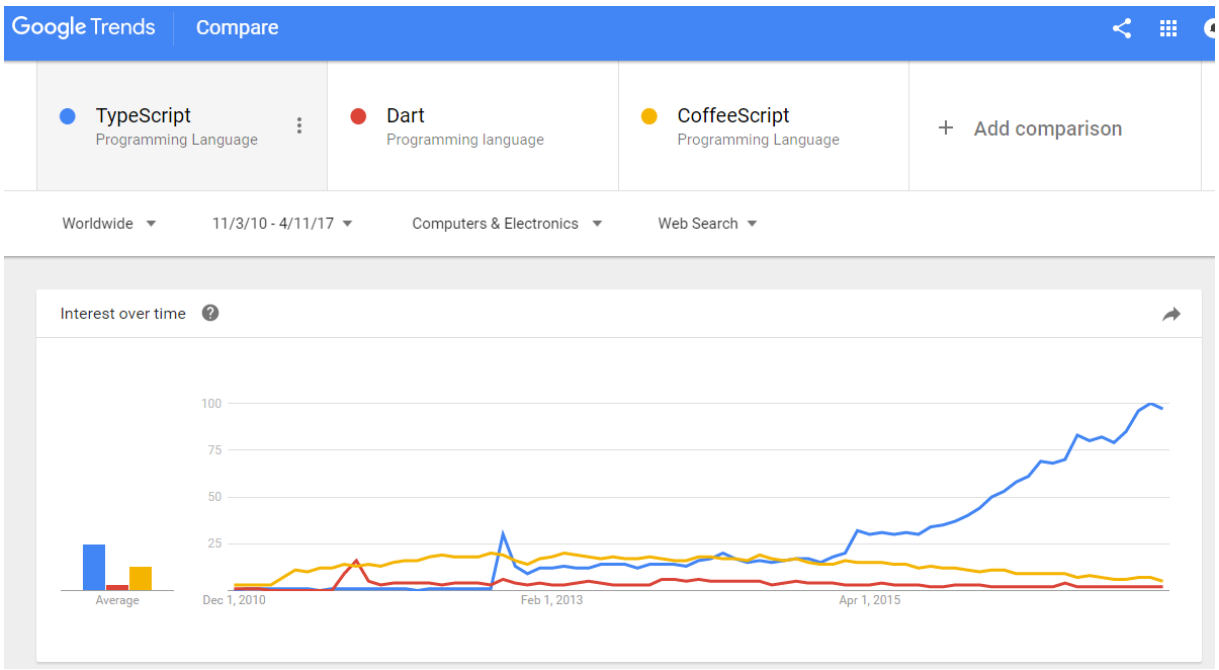
Z představených programovacích jazyků jsem zvolil jazyk TypeScript, protože přináší do jazyka JavaScript typovou kontrolu, která mi, jako programátorovi, který je zvyklý na staticky typované jazyky jako je Java, při práci s jazykem JavaScript poměrně dost chyběla. Typy není nutné definovat vždy, ovšem u větších projektů se hodí použít kompilační mód `--noImplicitAny`, který použití typů vynucuje tam, kde není možné odvodit typ z kontextu. U kratších skriptů bych řekl, že není použití tohoto módu nutné. Hlavní přínos jazyka TypeScript vidím v nástrojích, které jsou mocnější než podobné nástroje pro jazyku JavaScript. Refactoring a změny v rozsáhlém projektu jsou v jazyku TypeScript o dost snažší ve srovnáním s čistým JavaScript kódem.

Programovací jazyk Dart sice také umožňuje typovou kontrolu, ovšem spolupráce s externími knihovnami je v něm komplikovanější. Navíc kód, který je výstupem kompilátoru jazyka TypeScript je mnohem přehlednější, než výstup z kompilátoru jazyka Dart.

Pokud jde o jazyk CoffeeScript, tak ten je podle mého názoru spíše jenom syntakticky úhledněji zapsaný JavaScript a v jeho použití bych neviděl velký přínos. Řekl bych, že TypeScript toho programátorovi nabízí více.

Dalším neméně důležitým faktorem při vývoji v nějakém jazyku je velikost komunity kolem tohoto jazyku, od které se odvíjí možnost nalezení různých návodů, nástrojů a knihoven pro daný programovací jazyk. Díky definičním souborům a existenci repositáře Definitely typed[25], který umožňuje velmi snadné použití knihoven z JavaScript světa, se zachováním našeptávání ve vývojovém prostředí a statické typové kontroly během kompilace, je používání již hotových JavaScript knihoven velmi pohodlné.

Na následujícím obrázku z GoogleTrends je vidět, že TypeScript je poměrně oblíbený a jeho popularita, alespoň podle statistik společnosti Google, stoupá.



Obrázek 3.2 - Popularita jednotlivých jazyků[48]

Navíc známý framework Angular od společnosti Google je ve své druhé verzi vyvíjen právě v jazyku TypeScript. Vzhledem k tomu, že v tomto jazyku vývíjejí významné projekty společnosti typu Google a Microsoft, lze očekávat, že podpora jazyka TypeScript bude zajištěna i v dlouhodobém časovém horizontu.

Kapitola 4

Vývojové prostředí a nástroje

V této kapitole představím vývojové prostředí, nástroje pro automatizaci buildu a nástroje pro vývoj webových aplikací. Tato sada nástrojů se často označuje jako tzv. „development stack“. Uvedené nástroje jsem zkoušel na operačním systému Windows, ale vzhledem k tomu, že většina nástrojů je multiplatformní, lze tyto postupy, případně s drobnými úpravami, aplikovat i na ostatní operační systémy.

4.1 Node.js

Node.js je běhové prostředí pro JavaScript, které umožňuje spouštění různých serverových skriptů, nástrojů a aplikací. K interpretování skriptů v jazyce JavaScript node.js používá V8 engine od společnosti Google.

Node.js je založený na využívání modelu událostí a asynchronních I/O operací. Tudíž většina operací neblokuje vykonávání, ale výsledek je předán pomocí callbacku. Tento návrh se snaží usnadnit škálovatelnost webových aplikací, kde je hodně I/O operací. Na rozdíl od platforem jako Java nepodporuje více vláken, která sdílejí paměť. Pokud chce uživatel node.js využívat paralelismus při nějakém výpočtu, lze používat více procesů, které komunikují pomocí zpráv.

Node.js se kromě enginu V8 skládá z mnoha modulů, které umožňují přístup k systému souborů, k sítím (DNS, HTTP, TCP, TLS/SSL a UDP), práci s buffery, kryptografické funkce, data streamy a další.

Právě v běhovém prostředí node.js je spouštěno mnoho nástrojů a kompilátorů ze světa jazyka JavaScript od TypeScript a CoffeeScript kompilátorů přes buildovací nástroje Gulp a Grut po nástroj uglify pro minifikaci.

4.2 Vývojové prostředí

TypeScript je možné editovat v mnoha editorech. Já zvolil jsem vývojové prostředí **Visual Studio Code**[4] (dále VS Code). Jde o multiplatformní vývojové prostředí, které lze provozovat na operačních systémech Windows, Mac OS i Linux. Visual Studio Code je stejně jako TypeScript pod správou společnosti Microsoft. Přitom programovací jazyk TypeScript i zdrojové kódy VS Code jsou šířeny pod open-source licencí. TypeScript je šířen pod Apache License 2.0 a VS Code pod MIT licencí.

[4] https://code.visualstudio.com/

Tudíž je možné najít podporu jazyka TypeScript i ve vývojových prostředích, která nejsou pod kontrolou společnosti Microsoft.

Visual Studio Code podporuje interaktivní ladění (debugging), má vestavěnou podporu verzovacího systému GIT[5], zvýrazňování syntaxe, našeptávání kódu i refactoring. Pro JavaScript a TypeScript (za použití source maps) je v node.js dostupné debuggování. Dále je dostupný doplněk pro debuggování kódu, který běží v prohlížeči Google Chrome[6]. Google Chrome podporuje source mapping, tudíž při debuggování ukazuje přímo kód v jazyce TypeScript a ne výsledný JavaScript. Díky tomu je možné provádět debugging přímo v prohlížeči Chrome. Řekl bych, že krokování přímo v prohlížeči Chrome je snazší, protože není potřeba žádná konfigurace a vývojářské nástroje prohlížeče Chrome jsou na vysoké úrovni. Mojí zkušenost s VS Code bych shrnul tak, že jsem s VS Code poměrně spokojený a používám ho většinu času, co pracuji s jazykem TypeScript.

TypeScript má podporu i v jiných vývojových prostředích. Lze použít komerční vývojová prostředí jako WebStorm od JetBrains, Visual Studio od společnosti Microsoft, která také mají podporu pro jazyk TypeScript. Existují plugins do NetBeans i Eclipse. Ale zrovna plugin do NetBeans se mi zdál poměrně pomalý a během editování kódu, zvýrazňování syntaxe hrálo všemi barvami a refactoring nebyl mnohdy úspěšný, tudíž bych ho nedoporučil, je použitelný, ale programování ve VS Code bylo z mého pohledu příjemnější.

Další možností je editor Atom, ale VS Code mi vyhovuje více. P/odle mého názoru je předností VS Code je jeho rychlost. Přestože jsou obě prostředí založena na frameworku Electron, který se dříve se jmenoval Atom shell, je běh VS Code pocitově plynulejší.

4.3 Linting

Linting je kontrola správnosti zdrojového kódu a hledání potencionálních chyb. V jazyce JavaScript se používají různé nástroje, které poskytují varování o potencionálně chybném kódu. Ovšem kvůli dynamické povaze JavaScriptu nejsou výsledky tak spolehlivé jako u kompilátorů staticky typovaných jazyků. I ve srovnání například s jazykem TypeScript a jeho kompilátorem nejsou tyto nástroje tak mocné. Zástupci této třídy pomocných nástrojů jsou JSLint[7], JSHint[8] a nejnovější z nich ESLint[9].

Tento typ analytického nástroje existuje i pro jazyk TypeScript, jmenuje se TSLint[10], ale vhledem k tomu, že TypeScript kompilátor tsc, provádí mnoho typových kontrol během kompilace, tak jsem neměl potřebu tento nástroj využívat. Ovšem existuje přímo doplněk TSLint[11] do Visual Studia Code, který linting zdrojového kódu v jazyce TypeScript ve VS Code obstarává.

---

[5] https://code.visualstudio.com/docs/editor/versioncontrol

[6] https://github.com/Microsoft/vscode-chrome-debug

[7] JSLint, http://www.jslint.com/

[8] JSHint, http://jshint.com/

[9] ESLint, http://eslint.org/

[10] TSLint, https://palantir.github.io/tslint/

[11] TSLint doplněk do VS Code, https://marketplace.visualstudio.com/items?itemName=eg2.tslint

4.4 Minification

Minifikace, český ekvivalent by mohl být „minimalizace“, je proces odstranění nadbytečných prvků ze zdrojového kódu bez změny funkcionality kódu. Obvykle jde o bílé znaky, nové řádky a komentáře. Některé nástroje umožňují přejmenovat i lokální proměnné v JavaScript kódu, aby ušetřili ještě více místa.

Minifikace je široce používaná při distribuci aplikací vyvinutých v jazyce JavaScript. Zdrojové kódy velkých JavaScript aplikací mohou být poměrně veliké, například knihovna OpenLayers ve verzi 3.20.1 má 2,48 MB, což může výrazně ovlivnit rychlost spuštění webové aplikace založené na této knihovně. Minifikovaná verze té samé knihovny má 492 kB.

Zástupci této kategorie nástrojů jsou uglify-js, YUI Compressor a Google Closure Compiler. Například knihovna OpenLayers, která je představena dále v textu a slouží pro zobrazení webových map, používá Google Closure Compiler.

Closure compiler[12] může jít v minifikaci ještě dále, v takzvaném „Advanced“ módu přejmenovává všechna jména všech funkcí, atributů a proměnných, pokud nejsou dány samotnou běhovou platformou, například prohlížečem. Tudíž celá aplikace musí být zkompilována v jednom běhu, nebo je nutné definovat globální proměnné a funkce, které slouží ke komunikaci s externí knihovnou, pomocí takzvaných „Externs“. Naopak pokud je potřeba, aby nějaká globální proměnná nebyla přejmenována z důvodu toho, že by s její pomocí měla knihovna, kterou právě minimalizujeme, nabízet svoje rozhraní, pak je nutné tuto globální proměnou uvést v tzv. „Exports“.

4.5 Automatizace buildu v jazyce JavaScript

Během vývoje v jazyce JavaScript a v jazycích, které se do jazyka JavaScript kompilují, se vyskytuje velké množství úloh, které jde automatizovat, například:

- Kompilování zdrojových kódů v jazyce TypeScript případně v ES6 nebo v nějakém jiném alternativním jazyce do kódu v ES5, případně ES3, který je pak interpretován v prohlížečích.
- Minifikace zdrojových kódů pro zmenšení velikosti souborů.
- Kompilace SASS a LESS do CSS.

Tyto úlohy můžeme pouštět pomocí příkazové řádky, nebo je pouštět pomocí dávkových nebo bashových souborů. Případně můžeme zvolit některý z automatizovaných buildovacích nástrojů, jako jsou například dále zmíněné nástroje Gulp a Grunt.

4.5.1 Gulp

Gulp je streamovací build systém, který je postavený nad datovými proudy (streams) v node.js. To umožňuje většinu manipulace provádět v paměti a minimalizovat počet přístupů na disk. Zápis na disk je v ideálním případě proveden, až když je hotový konečný výsledek. Systém Gulp inklinuje k programovacímu přístupu, v porovnání se systémem Grunt, který je více závislý na konfiguraci.

V níže uvedeném příkladu je vidět konfigurace v buildovacím nástroje Gulp, která vezme zdrojový soubor, minifikuje ho pomocí nástroje Uglify a vytvoří source maps.

[12] Closure Compiler, https://developers.google.com/closure/compiler/

Příklad 4.1 – Konfigurace Gulp

```
var gulp  = require('gulp'),
    gutil = require('gulp-util');

gulp.task('build-js', function() {
  return gulp.src('source/app.js')
    .pipe(sourcemaps.init())
      .pipe(uglify())
    .pipe(sourcemaps.write())
    .pipe(gulp.dest('public/assets/javascript'));
});
```

4.5.2 Grunt

Grunt je starší buildovací nástroj než Gulp, místo streamů používá dočasné soubory. Každý task v buildovacím systému Grunt je pole konfigurací různých pluginů. Tyto plugins jsou jednoduše vykonávány s danými konfiguracemi. Plugins jsou vykonávány jeden po druhém, není zde žádné navazováni pomocí streamů jako u systému Gulp. Existují tisíce plug-ins, které umožňují automatizovaně provádět časté úkoly.

Příklad 4.2 – Konfigurace Grunt

```
grunt.initConfig({
  uglify: {
    my_target: {
      files: {
        'dest/output.min.js': ['src/input1.js', 'src/input2.js']
      }
    }
  }
});
```

V následující tabulce jsou uvedeny některé plugins pro buildovací systém Grunt, další plugins lze nalézt v kompletním seznamu na https://gruntjs.com/plugins.

Tabulka 4.1 – Přehled plugins pro build systém Grunt

Plugin	Popis
contrib-jshint	Validuje soubory pomocí nástroje JSHint
contrib-uglify	Minifikuje JS soubory pomocí nástroje UglifyJS
contrib-watch	Spustí nějaký task, kdykoli jsou sledované soubory změněny
contrib-clean	Smaže definované soubory, případně vyčistí definované složky
contrib-copy	Kopíruje definované soubory nebo složky
contrib-concat	Spojí více souborů do jednoho
contrib-cssmin	Minifikuje CSS soubory

contrib-less	Kompiluje LESS soubory do CSS souborů
contrib-imagemin	Minifikuje PNG, JPG, a GIF obrázky
contrib-compass	Kompiluje SASS do CSS použitím nástroje Compass
contrib-htmlmin	Minifikuje HTML soubory

4.6 Dynamické jazyky pro tvorbu stylesheetů

Less a **Sass** jsou programovací jazyky, ve kterých se definuje styl dokumentu, jde o jakýsi syntaktický cukr nad základními CSS soubory. Soubory napsané v těchto jazycích jsou potom přeloženy do CSS souborů.

Pokud naše webová aplikace využívá větší množství kaskádových stylů, tak se zanedlouho tyto styly stávají velmi nepřehledné. Proto vznikly nástroje jako je Less a Sass, které slouží ke zpřehlednění kaskádových stylů a k odstranění opakování.

Největší rozdíl mezi Less a Sass je, že Less je založené na jazyku JavaScript, tudíž je velmi snadné pustit překlad v prostředí Node.js. Alternativou kompilace před nasazením webové aplikace v produkci je překlad Less souborů přímo v cílovém prohlížeči, viz příklad níže. Naproti tomu Sass je založený na Ruby, což usnadňuje jeho nasazení do prostředí Ruby on Rails. **Zaměřím se na dynamické kaskádové styly v jazyku Less** vzhledem k tomu, že development stack představený v této práci je založený na jazyku JavaScript a Node.js.

Příklad 4.3 – Překlad Less souborů přímo v prohlížeči

```
<link rel="stylesheet/less" type="text/css" href="styles.less">
<script src="less.js" type="text/javascript"></script>
```

Díky nástroji Less je možné v kaskádových stylech používat proměnné a tím se vyhnout neustálému kopírování a zároveň se tím může usnadnit provádění změn, když je jedna barva na jediném místě, pak je změna barevného stylu aplikace mnohem snazší. Less používá před proměnou symbol @, naproti tomu Sass používá symbol $. Příklad definice proměnné v Less je vidět v níže uvedeném příkladu.

Příklad 4.4 – Proměnné v Less

```
/* Less */
@modra: #00c;
@svetle_modra: @blue + #333;

#header {
  color: @modra;
}
h2 {
  color: @modra;
}

/* Vygenerované CSS */
```

```
#header {
  color: #00c;
}
h2 {
  color: #00c;
}
```

Další vymožeností je možnost používat jeden styl v různých třídách, čímž se zabrání zbytečnému kopírování opakovaně používaných částí kaskádových stylů. Tomuto způsobu opakovaného používání stylů se říká „Mixins“, příklad použití v Less je na níže uvedeném příkladu, kde máme definovanou třídu `okraj` a znovu ji používáme v jiné třídě `clanek`, čímž se vyhneme opakování, případně nutnosti definovat u jednoho elementu v HTML mnoho tříd.

Příklad 4.5 – Mixins v Less

```
.okraj{
  border-style: ridge;
  border-radius: 10px;
  border-width: 2px;
}

.clanek{
  background: #efe;
  .okraj;
}
```

Navíc je možné používat tzv. „Parametric mixins“, viz příklad níže. Tento druh mixins umožňuje definovat výchozí hodnotu parametru, která se použije v případě, že je mixins použitý v následující formě „`.parametrickeZaobleni`“. Uživatel ovšem může stejný mixins použít i s parametrem „`.parametrickeZaobleni(10px)`“.

Příklad 4.6 – Parametric mixins v Less

```
.parametrickeZaobleni( @zaobleni: 3px ) {
  -webkit-border-radius: @zaobleni;
  -moz-border-radius: @zaobleni;
  border-radius: @zaobleni;
}
```

Níže uvedený příklad ukazuje, jak lze v dynamickém stylovacím jazyku Less definovat zanoření pomocí složených závorek v podobném stylu, jako je definováno zanoření bloků v jazycích z rodiny jazyků odvozených z jazyka C.

Příklad 4.7 – Zanoření CSS vs Less

```
/* Zanoření v CSS */
#site-body .post .post-header h2
```

```
/* Zanoření v Less */
#site-body { …

    .post { …

        .post-header { …

            h2 { … }

            a { …

                &:visited { … }
                &:hover { … }
            }
        }
    }
}
```

Jazyk Less umožňuje i použití operátorů a funkcí. Less podporuje sčítání, odčítání, násobení a dělení barev a různých dalších číselných hodnot stejných typů. Operace s různými jednotkami nejsou v less podporovány, například nelze sčítat palce (in) a centimetry (cm). Naproti tomu Sass má převod jednotek vestavěný.

Příklad 4.8 – Operátory v Less

```
/* Operátory v Less */
@okraj: 1px;
@zakladni-barva: #111;
@cervena:        #842210;

#header {
  color: @zakladni-barva* 3;
  border-left: @okraj;
  border-right: @okraj * 3;
}
#footer {
  color: @zakladni-barva + #003300;
  border-color: desaturate(@cervena, 10%);
}

/* Vygenerované CSS */
#header {
  color: #333333;
  border-left: 1px;
  border-right: 3px;
}
```

```
#footer {
  color: #114411;
  border-color: #7d2717;
}
```

Less podporuje jak standardní CSS komentáře `/* CSS Komentář */` tak i jednořádkové `//Jednořádkový JS Komentář`. Rozdíl je v tom, že jednořádkové JS komentáře jsou při zpracování less souborů odstraněny a nejsou součástí výsledného CSS souboru. Naproti tomu standardní CSS komentáře zůstávají i ve výsledném CSS souboru.

4.7 Vybraný build nástroj a jazyk pro tvorbu stylesheetů

Z představených build nástrojů **bych doporučil Gulp.** Díky použití datových proudů je build s nástrojem Gulp rychlejší v porovnání s nástrojem Grunt. Gulp je více zaměřený na programovací přístup, konfigurace se píše kódem v jazyce JavaScript. Naproti tomu nástroj Grunt je založen na konfiguraci, která se podobá zápisu objektů ve formátu JSON. Mnohdy je konfigurace v Gulp kratší a přehlednější ve srovnání s konfigurací pro nástroj Grunt. Obzvláště ve větších projektech s komplikovaným buildem se konfigurační soubory pro systém Grunt mohou stát velmi nepřehledné.

Na druhou stranu oba dva nástroje slouží svému účelu velmi dobře, tudíž pokud už nějaký projekt používá nástroj Grunt a není zde problém s příliš nepřehlednou konfigurací buildu, tak není důvod k přechodu. Naopak díky tomu, že nástroj Grunt je starší, tak je pro něj dostupné větší množství plugins ve srovnání nástrojem Gulp.

Pokud projekt obsahuje pouze málo kaskádových stylů, tak je možné používat přímo CSS soubory. Ovšem když začne množství stylů růst, tak bych **doporučil** přejít na **nástroj Less**. Styly v Less jsou přehlednější a lépe strukturované. Navíc díky tomu, že je Less postavený na jazyku JavaScript, tak lépe zapadá do development stack popsaného v této diplomové práci a je ho možné velmi snadno kompilovat rovnou při buildu webové aplikace pomocí Node.js (Less podporují oba build nástroje Grunt i Gulp), případně až v cílovém prohlížeči.

Kapitola 5

JavaScript knihovny pro zobrazení map

V následující kapitole se budu zabývat různými knihovnami pro zobrazení map ve webových prohlížečích. Při porovnání se zaměřím na zobrazení rastrových dlaždic s mapovými daty. Dále by daná knihovna měla podporovat zobrazení vektorů, které jsou zobrazeny ve vrstvě nad nimi.

V současné době se pro většinu použití prosadilo řešení ve stylu Google maps. Google se svým řešením přišel poprvé v roce 2005. Velmi podobné řešení používají i OpenStreetMaps, Bing maps a další mapy zobrazené ve webových prohlížečích.

Při použití tohoto řešení má dlaždice typicky rozměr 256x256 pixelů a povrch země je na nich zobrazen v projekci WebMercator (jinak také nazývaná Google Web Mercator, Spherical Mercator, WGS 84 Web Mercator, WGS 84/Pseudo-Mercator nebo oficiálně EPSG:3857)[36]. Projekce WebMercator pokrývá rozsah 180° západní délky až 180° východní délky a 85.0511° severní šířky až 85.0511° jižní šířky.

Nejnižším přiblížením je 0, při přiblížení 0 se celý svět vejde do jediné dlaždice. Při každém dalším přiblížení je plocha, která byla původně reprezentována jedinou dlaždicí, reprezentována čtyřmi podrobnějšími dlaždicemi. Tudíž při přiblížení 1 je svět reprezentován 4 dlaždicemi, při přiblížení 2 je zapotřebí 16 dlaždic, více v následující tabulce[35].

Tabulka 5.1 – Počty dlaždic v OSM

Zoom	Počet dlaždic		Velikost dlaždic ve stupních (d×š)
n	$2^n \times 2^n$	2^{2n}	$\left(\frac{360}{2^n}\right)^\circ$ × různé
0	1 pro celý svět	1	360° × 170,1022°
1	2 × 2	4	180° × 85,0511°
2	4 × 4	16	90° × různé
3	8 × 8	64	45° × různé
4	16 × 16	256	22,5° × různé
...			

Tabulka 5.1 – Počty dlaždic v OSM

Zoom	Počet dlaždic		Velikost dlaždic ve stupních (d×š)
12	4096 × 4096	16 777 216	0.0879° × různé
...			
16	65 536 × 65 536	4 294 967 296	≈ 0,005493 × různé
17	131 072 × 131 072	17 179 869 184	≈ 0,002746 × různé

Kromě výše zmíněných konvencí, které vychází z Google maps, se de facto standardem stalo použití systému XYZ. Kde Z reprezentuje přiblížení dlaždice a X a Y identifikují, o jakou dlaždici v rámci přiblížení se jedná. Dlaždice se většinou stahují pomocí RESTového API, kde má adresa následující schéma „`http://.../Z/X/Y.png`“. Dlaždice jsou většinou ve formátu PNG, případně JPG.

5.1 OpenLayers

OpenLayers je open-source knihovna pro zobrazení map. Jde o poměrně velkou knihovnu, která zabírá 492 KB (platí pro minifikovanou verzi 3.20.1, neminifikována verze, která se používá při debuggování, má dokonce 2,48 MB). Openlayers obsahuje mnoho funkcí už v základu. Je postavená na kompilátoru Google Closure, který provádí typovou kontrolu podobným stylem jako TypeScript. Pomocí Google Closure kompilátoru je možné si udělat vlastní odlehčený build, který obsahuje pouze části, které daný uživatel této knihovny využívá.

Knihovna OpenLayers je poměrně komplexní a je vhodná pro komplikované GIS-like aplikace. Podporuje mnoho formátů vektorových data jako KML, GML, GeoJSON a mnoha dalších[87]. Knihovna podporuje i GeoRSS a datové zdroje v OGC (Open Geospatial Consortium) standardech Web Map Service (WMS) nebo Web Feature Service (WFS). Souřadnice se do této knihovny zadávají ve formátu `[zeměpisná délka, šířka]`.

Knihovna OpenLayers sama o sobě podporuje pouze ploché zobrazení map, ale je možné použít knihovnu Ol-Celsium[89], která integruje knihovny OpenLayers a Celsium, a umožňuje zobrazovat mapy projektované na 3D model země. Přestože je toto zobrazení velmi efektní, tak knihovna Ol-Celsium ještě není příliš vyspělá (2017) a 3D zobrazení je poměrně náročné na výkon počítače, na kterém běží webový prohlížeč.

Příklad 5.1 – Zobrazení mapy s knihovnou OpenLayers

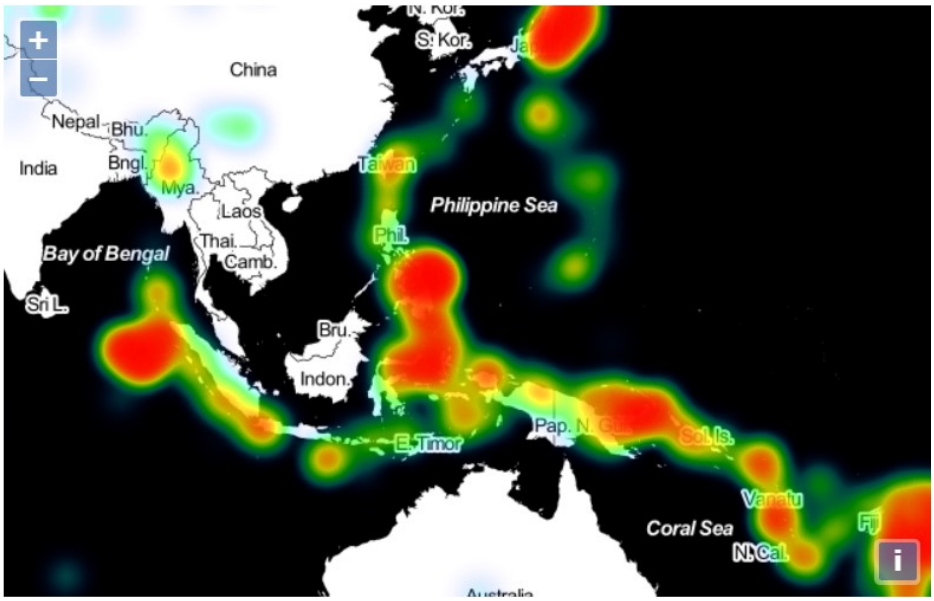
```
//Html
<link rel="stylesheet"
href="https://openlayers.org/en/v4.1.0/css/ol.css" type="text/css">
<script src="https://openlayers.org/en/v4.1.0/build/ol.js"></script>
<div id="map" class="map" tabindex="0"></div>

//JavaScript
var map = new ol.Map({
  layers: [
```

```
    new ol.layer.Tile({
      source: new ol.source.OSM()
    })
  ],
  target: 'map',
  controls: ol.control.defaults({
    attributionOptions: ({
      collapsible: false
    })
  }),
  view: new ol.View({
    center: [0, 0],
    zoom: 2
  })
});
```

Ve výše uvedeném příkladu je vidět jak se do OpenLayers zadává pole `layers`, které obsahuje jednotlivé vrstvy.

Knihovna OpenLayers podporuje například následující typy vrstev:

- `ol.layer.Tile` – Zobrazuje mapové dlaždice, které jsou předem vyrenderované na serveru.
- `ol.layer.Vector` – Zobrazuje vektorová data, která jsou renderovaná přímo v prohlížeči.
- `ol.layer.VectorTile` – Funguje a používá se podobně jako předchozí druh vrstvy `ol.layer.Vector`, s tím rozdílem, že je zde buffer, do kterého se jednotlivé vektory předrenderovávají. To umožňuje plynulejší pohyb mapy a změnu přiblížení, cenou za toto zrychlení je chvilková pixelizace během přibližování.
- `ol.layer.Heatmap` – Vektorová data jsou agregována do barevné vrstvy, kde platí například čím červenější, tím více bodů je v daném místě. Na obrázku níže je vidět Heatmap, která zobrazuje zemětřesení v západním pacifickém regionu.
- `ol.layer.Image` – Zobrazuje obrázek na určité místo na mapě.



Obrázek 5.1 – Příklad Heatmap v OpenLayers[64]

Další položkou, kterou je nutné uvést při inicializaci mapy je `target`, který obsahuje buď `id` elementu, do kterého má být mapa vykreslena nebo element samotný.

Položka `view` určuje výchozí oblast, která bude při spuštění webové aplikace nebo při načtení webové stránky zobrazena. Výchozí view je určen pomocí souřadnic středu mapy, které jsou zadány jako pole `[zeměpisná délka, zeměpisná šířka]` a pomocí výchozího přiblížení.

Položka `controls` určuje, jaké ovládací prvky mají být zobrazeny. Je možné používat předpřipravené ovládací prvky nebo si definovat vlastní.

5.2 Leaflet

Leaflet je poměrně malá knihovna pro zobrazení map. Zabírá výrazně méně než OpenLayers, 142 KB v případě minifikované verze 1.0.3 a 373 KB v případě neminifikované verze. Na webových stránkách knihovny Leaflet je dokonce uvedeno 38 KB, to je ovšem velikost knihovny komprimované gzipem. Knihovna Leaflet je přátelská jak k začátečníkům, tak i k mobilním zařízením. Leaflet má v základu poměrně málo funkcí, zato nabízí stovky pluginů[83]. Na druhou stranu velké množství pluginů přináší problémy s roztříštěnou dokumentací a nekompatibilitou pluginů se staršími nebo novějšími verzemi knihovny[44].

Souřadnice se do knihovny Leaflet zadávají ve formátu zeměpisná šířka/délka. Jde o knihovnu, která je vhodná pro jednoduché webové mapy. Na rozdíl od OpenLayers Leaflet nemá podporu pro projekci map na 3D model země.

Příklad 5.2 – Zobrazení mapy s knihovnou Leaflet

```
//HTML
<div id="mapid" style="width: 600px; height: 400px; position:
relative;" class="leaflet-container leaflet-touch leaflet-retina
leaflet-fade-anim leaflet-grab leaflet-touch-drag leaflet-touch-
zoom" tabindex="0"></div>
```

```
//JavaScript
var mymap = L.map('mapid').setView([51.505, -0.09], 13);

L.tileLayer('http://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png',
{
        maxZoom: 18,
        attribution: 'Map data © <a
href="http://openstreetmap.org">OpenStreetMap</a> contributors',
}).addTo(mymap);
```

5.3 Google Maps API

Podporuje 3D zobrazení zeměkoule ve stylu glóbusu, jde o velmi vyspělé řešení, které zvládá zobrazovat ortodromy už v základu, tj. při zobrazování delších spojnic dvou bodů na zemském povrchu se projevuje zakřivení země.

Na druhou stranu nejde o open-source řešení, které je navíc plně závislé na společnosti Google. Což je naprosto nevhodné, pokud například počítáme, že by k naší mapové aplikaci měli mít přístup lidé z intranetu nějaké společnosti, kteří nemusí mít přístup na internet.

5.4 Další možnosti

ModestMaps je velmi minimalistická knihovna pro zobrazení map. Její minifikovaná verze zabírá pouhých 35 KB. Přes svoji minimalistickou velikost má tento projekt zabudovanou podporu pro zobrazování ortodrom, tj. při zobrazování cest na zemském povrchu se projevuje zakřivení. Projekt není příliš aktivní, poslední vydaná verze je 1.0.0 beta z roku 2011, na githubu sice nějaká aktivita je[84], ale jde spíše o skomírající projekt, proto bych jeho použití nedoporučil. V případě nutnosti použití lehkého řešení bych spíše zvolil knihovnu Leaflet.

Další alternativou by mohlo ještě být ESRI JavaScriptové API, což je velmi vyspělé řešení, ale na druhou stranu není open-source a je velmi závislé na services od ESRI, což by přinášelo podobné problémy jako Google Maps API.

5.5 Porovnání mapových knihoven

Knihovna OpenLayers poskytuje velké množství funkcí v ucelené knihovně, na rozdíl od knihovny Leaflets. Tím je snížena nutnost použití mnoha plug-ins pro doplnění pokročilejší funkcionality.

GoogleMaps jsou příliš svázané s jedním poskytovatelem, v tomto případě Google, a proto nesplňují podmínku, aby daná aplikace byla schopná běhu i v interní síti bez přístupu k internetu.

Knihovna ModestMaps je příliš minimalistická a ESRI jsou podobně jako GoogleMaps vázané na jediného poskytovatele mapových podkladů a závislé na připojení k internetu.

Tabulka 5.2 – Porovnání mapových knihoven

Knihovna	Velikost (před minifikací)	Licence	Klady	Zápory
OpenLayers	492 KB (2,48 MB)	2-clause BSD	• Vše v jednom balíčku, kompatibilní • Mnoho funkcí	• Velká knihovna • Komplikovanější pro jednoduché aplikace
Leaflets	142 KB (373 KB)	2-clause BSD	• Mnoho doplňků • Poměrně malé	• Nekompatibilita doplňků a různých verzí knihovny
ModestMaps	35 KB (100 KB)	2-clause BSD	• Minimální velikost	• Nevyvíjí se • Málo funkcí
GoogleMaps	cca 250 KB[13]	proprietární	• Mnoho funkcí • Kvalitní mapové podklady od Google	• Nefunguje bez internetu • Při větším počtu dlaždic nutno platit za provoz

Pro webovou mapovou GIS-like aplikaci, o které se jedná v této diplomové práci, bych doporučil knihovnu **Open Layers**, protože není závislá na konkrétním dodavateli mapových dlaždic a aplikace postavené na této knihovně mohou fungovat i v prostředí intranetu, kde není dostupné připojení na internet.

Knihovna Open Layers je poměrně velká a ucelená, obsahuje vestavěnou podporu projekce EPSG:4326, EPSG:3857 a podporuje knihovnu proj4js, podporuje mnoho formátů mapových dat (KML, GML, GeoJSON a mnoho dalších[87]) a není příliš závislá na externích plugins. Většinu funkcí obsahuje už v základu. To umožňuje snížit počet závislostí, čímž se zjednodušuje jejich správa a usnadňuje vývoj. Díky malému počtu závislostí lze poměrně snadno provádět update na novější verzi knihovny, beze strachu z toho, že nějaký důležitý plugin nebude s novější verzí fungovat.

Open Layers podporuje mnoho zdrojů map, takže by nebyl problém v budoucnu pro uživatele, kteří mají přístup na internet, přidat i možnost zvolit si mapové podklady třetích stran, například z Google maps nebo z mapy.cz.

[13] Záleží na konkrétním použití, daná velikost se vztahuje na následující příklad: https://google-developers.appspot.com/maps/documentation/javascript/examples/full/map-simple, citováno 19. 5. 2017

Kapitola 6

Testování webových aplikací

Testování při vývoji webových aplikací je neméně důležité než při vývoji desktopových a serverových aplikací. Naopak, kvůli možnosti nekompatibility mezi různými prohlížeči a mezi různými verzemi prohlížečů, je nutné testovat o to více. V této kapitole si představíme různé frameworky a knihovny pro testování webových aplikací.

6.1 Úvod do testování

V této kapitole představím několik pojmů vztahujících se k testování aplikací, některé tyto pojmy budou použity dále v textu s předpokladem, že čtenář dané pojmy už zná.

6.1.1 Unit testy

Unit testy jsou zaměřeny na nejmenší testovatelnou jednotku zdrojového kódu. Jednotkou se obvykle myslí jedna konkrétní třída nebo skupina tříd. Ovšem některé testy mohou mít užší zaměření, například mohou testovat pouze určitou skupinu metod nebo konkrétní metodu. Naopak některé unit testy se mohou zaměřovat na celé programy, například v případě, že se systém skládá z mnoha velmi jednoduchých programů.

6.1.2 Integrační testy

Integrační testování se provádí po unit testech a slouží k ověření spolupráce jednotlivých modulů a komponent. Integrační testování může být prováděno iterativně, tj. integrace modulů a komponent probíhá postupně, nebo může jít o big-bang přístup, kde je většina modulů integrována v jediném kroku.

Iterativní integrace modulů může probíhat od spodu, tzv. „bottom-up“ testování, kde nejnižší úrovně integrace jsou testovány jako první, například integrace párů modulů a postupně se přidávají další moduly.

Opakem tohoto přístupu je „top-down“ testování, kde se prvně testují větší celky a pak se postupuje stromovým způsobem k nižším úrovním. Kombinací metod bottom-up a top-down je tzv. sendvičové testování.

Big-bang, integrace modulů v jediném kroku, šetří čas, ale integrační proces může být poměrně komplikovaný a při chybách nemusí být zřejmé, z jakého modulu chyby pocházejí.

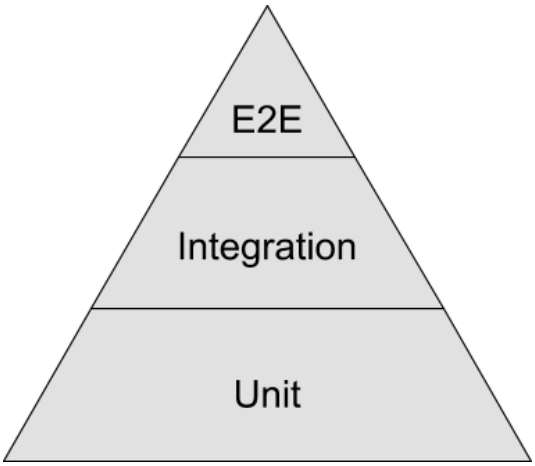
6.1.3 Funkcionální (GUI) testování/end-to-end

Většina uživatelů aplikací během interakce s aplikací tráví všechen čas s uživatelským rozhraním a právě proto je vhodné jej při testování neopomíjet. Unit testy testují back-end, mohou otestovat chování jednotlivých tříd, ale neotestují samotnou interakci uživatelského rozhraní s uživatelem, od toho jsou tu UI testy, které přímo interagují s uživatelským rozhraním aplikace.

6.1.4 Pyramida testů

S vyšší úrovní testů roste čas exekuce těchto testů, náročnost údržby a křehkost těchto testů. Na níže uvedeném obrázku je vidět testovací pyramida, která znázorňuje ideální poměr unit, integračních a E2E testů.

Unit testů by mělo být nejvíce, protože je nejlevnější je udržovat a jejich běh bývá velmi rychlý. Počet testů vyšších úrovní by se měl postupně snižovat s tím, jak stoupají jejich náklady na údržbu, doba běhu a křehkost. Na druhou stranu je důležité mít přiměřené množství E2E testů, protože existují chyby, které odhalí pouze tyto a manuální testy.



Obrázek 6.1 – Pyramida automatizovaných testů[85]

6.1.5 SUT a DOC

- **SUT** znamená **System Under Test**[3], testovaný systém. Testovaný systém může být třída nebo celá aplikace.
- **DOC** znamená **Depended On Component**[3], komponenta na které SUT závisí, aby mohl vykonávat úkony během testování. Většinou má DOC stejnou granulitu jako SUT, tj. když SUT je třída, tak DOC většinou bývají také třídy.

6.1.6 Test double

Test double je obecný název, pod který se schovají různé typy pomocných objektů používaných při testování systémů, hlavně při unit testování.

Gerard Meszaros[3] definoval několik typů „Test Doubles“:

- **Test stub** – nahrazuje komponentu, na které závisí SUT (testovaný systém), poskytuje testovanému systému nepřímé vstupy.
    - **Responder** – poskytuje validní hodnoty.

    - **Saboter** – poskytuje nevalidní hodnoty, používá se k testování toho, jak se aplikace chová právě v případě, že dostane nevalidní odpověď od DOC (systém, na kterém závisí testovaný systém).
- **Mock object** – jde o objekt, který může sloužit k ověření nepřímých výstupů testovaného systému. Obvykle obsahuje i funkcionalitu stejně jako Test stub, protože musí také vracet nějaké hodnoty, ovšem důraz je zde kladen na otestování výstupu SUT.
- **Test spy** – jde o rozšířenou formu Test stub objektu, která kromě poskytování nepřímých vstupů ještě navíc zaznamenává nepřímé výstupy SUT, pro pozdější vyhodnocení. Jde o jakýsi test stub s možností nahrávání výsledků.
- **Fake object** – Pokud není reálný objekt dostupný, nebo je příliš pomalý, používá se fake object. Je to zjednodušená implementace, která nahrazuje DOC. Například místo opravdové databáze, lze během testu používat hash-mapu, která je v paměti, což může výrazně zrychlit vykonávání testů.
- **Dummy object** – Pokud testovaný objekt potřebuje nějaké objekty jako parametr, tak lze místo pravého objektu posílat právě dummy objekt. Který se tváří jako pravý objekt, ale ve skutečnosti je v něm implementováno pouze minimální zdání funkčnosti.

6.2 Unit testy při vývoji webové aplikace

Unit testy se v případě webových aplikací dělí na dvě základní skupiny:

- testy, co běží ve webovém prohlížeči
- testy, co běží mimo prohlížeč, například v node.js.

6.2.1 Jasmine

Jasmine je open-source testovací framework pro JavaScript. Cílem tohoto frameworku je, aby mohl běžet na všech platformách, které podporují JavaScript, snaží se být nezávislý na vývojových prostředích. Dále je zde kladen velký důraz na jednoduchou čitelnou syntaxi.

Kvůli nezávislosti na běhovém prostředí, Jasmine sám o sobě nemá nástroj pro pouštění testů z příkazové řádky. Testy se pouští například tak, že se otevře webová stránka ve webovém prohlížeči a na této stránce se testy spustí. Nebo je možné testy pouštět například pomocí dále zmíněného nástroje Karma.

Tento Framework podporuje asynchronní testování. Ve výchozím nastavení čeká Jasmine framework 5 sekund na výsledek, než je vyhozen timeout failure. Vlastní timeout je možné nastavit pomocí globální proměnné `jasmine.DEFAULT_TIMEOUT_INTERVAL`. Pokud chceme změnit timeout u určitého testu, lze si uložit původní timeout do proměnné daného testu a timeout změnit v `beforeEach` bloku a pak timeout nastavit zpět na původní hodnotu v `afterEach` bloku.

V níže uvedeném příkladu je vidět jednoduchý objekt Calculator, který umí pouze sčítat dvě čísla.

Příklad 6.1 – Jasmine testovaný JS program

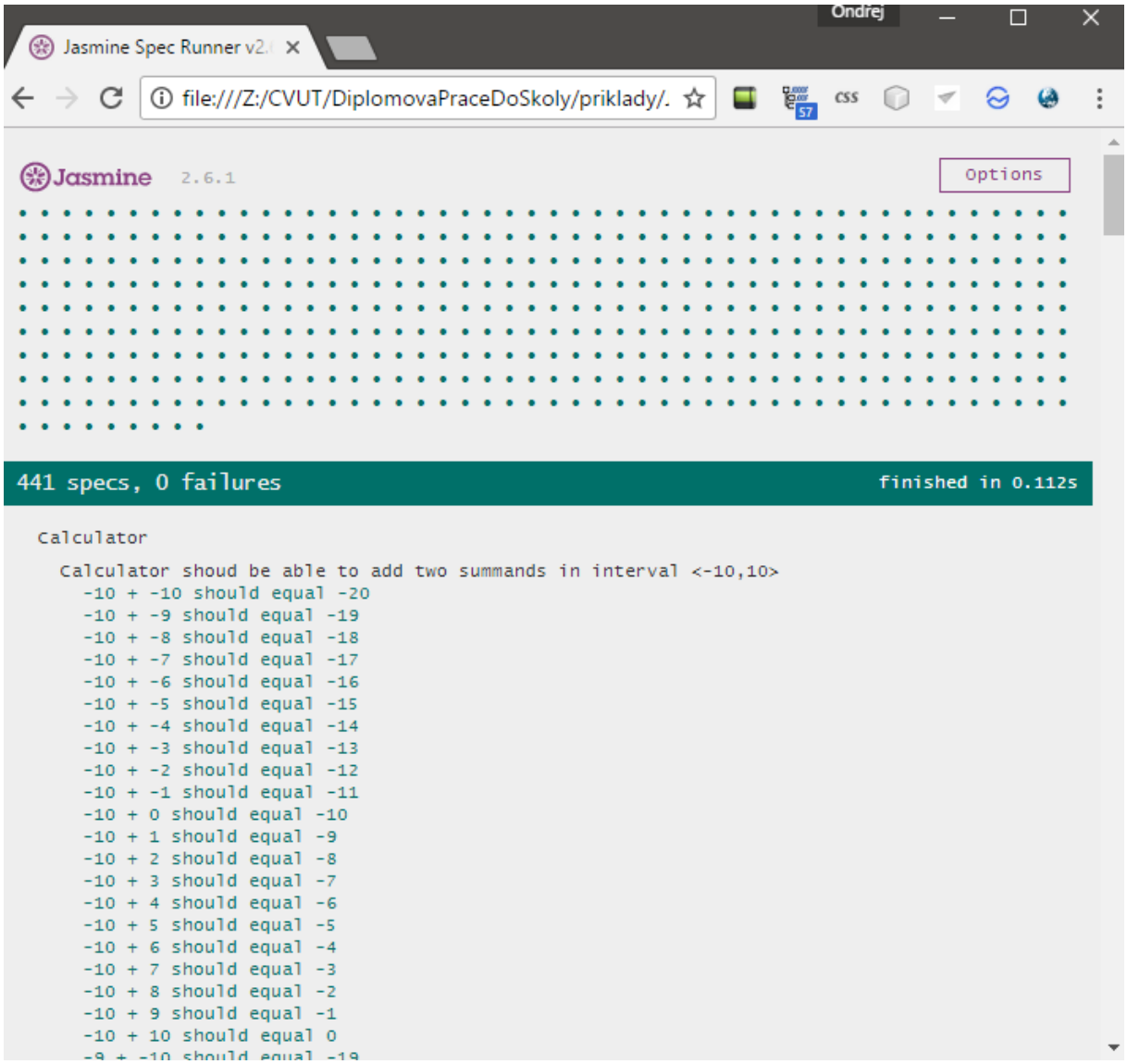
```
function Calculator() {}

Calculator.prototype.add = function(summand1, summand2) {
  return summand1 + summand2;
};
```

V dalším příkladu je test výše uvedeného programu, který testuje, zda program zvládá správně sčítat všechna čísla v intervalu <-10, 10>.

Příklad 6.2 – Jasmine test

```
describe("Calculator", function() {
   var calculator;

   beforeEach(function() {
      calculator = new Calculator();
   });

  describe("Calculator should be able to add two summands in
interval <-10,10>", function() {
    function testAdd(summand1, summand2, expectedResult){
        it(""+summand1+" + "+summand2+" should equal
"+expectedResult, function() {
            var result = calculator.add(summand1, summand2);

            expect(result).toEqual(expectedResult);
        });
    }//func testAdd

    for(var summand1=-10; summand1<=10; summand1++){
        for(var summand2=-10; summand2<=10; summand2++){
            var expectedResult = summand1+summand2;
            testAdd(summand1, summand2, expectedResult);
        }//for summand2
    }//for summand1
  });
});
```

Níže uvedený obrázek ukazuje způsob, jakým jsou zobrazeny výsledky ve frameworku Jasmine. V první sekci je výsledek testu vizualizován. Každý úspěšný test je reprezentován malým zeleným kolečkem . Naproti tomu neúspěšné testy jsou reprezentovány červeným křížkem . Dále následuje statistika, kolik `it` bloků bylo celkem definováno a kolik z nich bylo neúspěšných. Po statistice jsou stromovou formou uvedeny `declare` bloky a jejich `it` bloky. Texty `it` bloků je možné rozkliknout pro zjištění podrobností.



Obrázek 6.2 – Výsledek Jasmine testů

V případě, že při Jasmine testu vzniknou chyby, obsahuje Jasmine report dvě záložky. Jako výchozí je otevřena záložka „Failures“, která obsahuje seznam všech chyb s jejich popisy a stack-trace výpisy. Další záložka je „Spec list“, která se zobrazí i v případě, že se během testování nenarazí na žádné chyby. Tato záložka je zobrazena na výše uvedeném obrázku. Testy (`it` bloky), které prošly, jsou vykreslené zeleně. Ty, co neprošly, jsou naopak zobrazeny červeně.

6.2.2 Mocha.js

Mocha je testovací nástroj, který spouští testy v Node.js. Tento nástroj se ovládá z příkazové řádky a lze ho velmi dobře integrovat s IDE. Díky tomu, že testy běží v Node.js, je velmi pohodlně možné testy napsané v tomto nástroji spouštět jako součást build úlohy v budovacích nástrojích Gulp a Grunt.

Nevýhoda tohoto přístupu je v tom, že pomocí testů pouštěných v Node.js nelze odhalit různé problémy různých webových prohlížečů, které se na Node.js projevit nemusejí. Další nevýhodou je, že nelze používat API rozhraní, která jsou pouze v prohlížeči jako například DOM.

Testy napsané v Mocha.js lze pouštět i ve webovém prohlížeči, pak ovšem musí být pokaždé, když chce vývojář spustit testy, otevřena testovací webová stránka. Případně lze použít nástroj Karma, který je zmíněný dále v textu a umožňuje testovací webové stránky automatizovaně otvírat.

6.2.3 QUnit

QUnit je JavaScript testovací framework. Tento framework je používaný jQuery projektem k testování jQuery kódů a jejich plugins, ale umí testovat libovolnou JavaScript aplikaci. Kromě testování v prohlížeči, zvládne QUnit i testování JavaScript kódu na serverové straně v node.js.

Příklad 6.3 – Příklad testu ve frameworku QUnit[81]

```
//HTML
<html>
<head>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width">
  <title>QUnit Example</title>
  <link rel="stylesheet" href="https://code.jquery.com/qunit/qunit-
2.3.2.css">
</head>
<body>
  <div id="qunit"></div>
  <div id="qunit-fixture"></div>
  <script src="https://code.jquery.com/qunit/qunit-
2.3.2.js"></script>
  <script src="tests.js"></script>
</body>
</html>

//JavaScript
QUnit.test( "hello test", function( assert ) {
  assert.ok( 1 == "1", "Passed!" );
});
```

6.2.4 Karma

Karma je nástroj, který umožňuje spouštět JavaScript v několika prohlížečích. Nejde o testovací Framework, ani assert knihovnu. Karma pouze spustí http server, vygeneruje testovací HTML soubor pro testovací Framework jako je Jasmine, Mocha, QUnit a mnoho dalších. Seznam podporovaných testovacích frameworků je možné nalézt na npm pod klíčovým slovním spojením „karma-adapter“[14].

Výhodou karmy je, že umožňuje snadno automatizovaně spouštět různé testy v různých prohlížečích, ve kterých chceme dané testy pouštět. Alternativou k automatickému pouštění prohlížečů je zapnout prohlížeč a otevřít adresu, na které běží Karma server, obvykle to je

[14] Karma podporované frameworky: https://www.npmjs.com/browse/keyword/karma-adapter

`http://localhost:9876/`. Případně může Karma sledovat určitou množinu souborů a pustit testy při každé změně některého z těchto souborů.

6.3 GUI testovací frameworky/aplikace/E2E testy

V této části se zaměřím na frameworky a aplikace, které aplikaci testují přímo pomocí interakce s uživatelským rozhraním, tzv. GUI/UI/E2E testy.

6.3.1 TestComplete

TestComplete je platforma pro GUI testování, která je rozdělena do třech modulů, „Desktop“, „Web“ a „Mobile“. Osobně mám poměrně rozsáhlou zkušenost s moduly Web a Desktop.

Desktop modul je určen pro testování aplikací na systému Windows. TestComplete podporuje mnoho GUI frameworků. Tudíž je TestComplete schopen ovládat většinu komponent uživatelského rozhraní. Do výčtu podporovaných technologií patří Standard Windows Controls, Java Controls, JavaFX Controls, Microsoft Controls, Qt Controls, VCL a CLX Controls, WPF Controls a mnoho dalších. Ve srovnání se zdarma dostupnými řešeními jako je AutoIt, nebo jeho open-source variantou AutoHotkeys, kteří jsou omezení jen na nejzákladnější GUI komponenty, jde o velmi schopný automatizační software, který je schopný ovládat širokou škálu GUI komponent.

TestComplete podporuje dva základní typy testů:

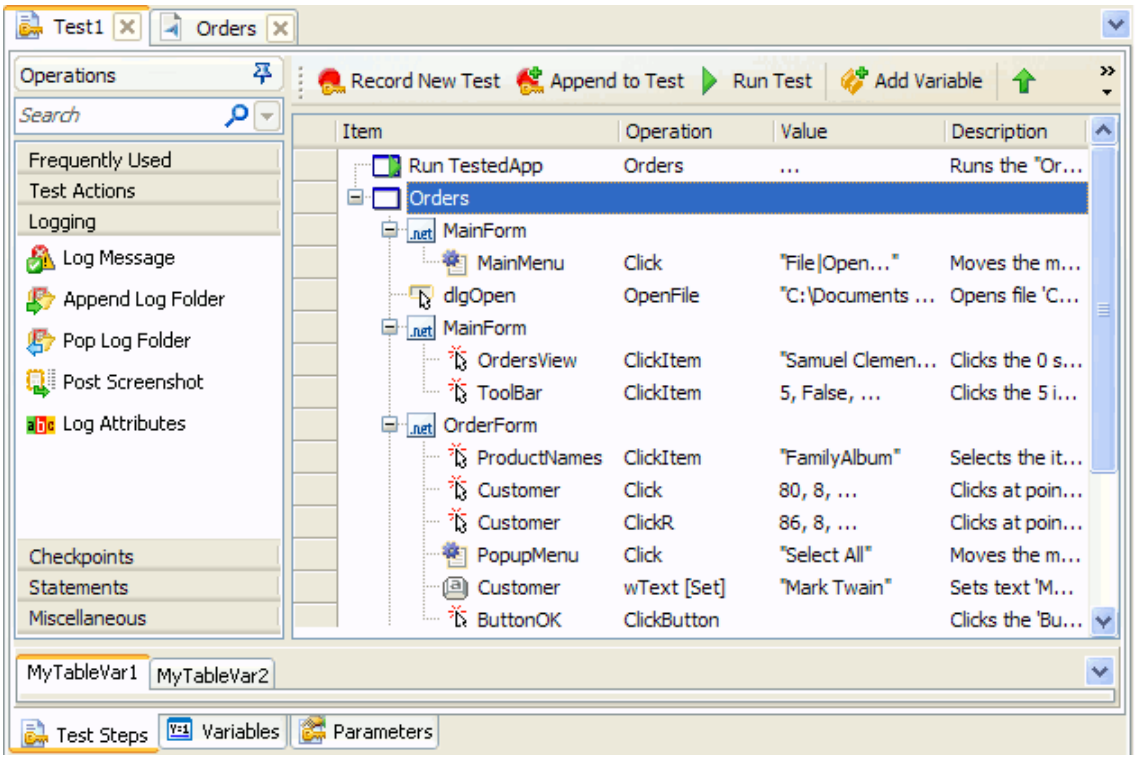
- **Keyword testy** – test je definován jako posloupnost kroků editoru, kroky jsou zobrazeny jako stromová struktura
- **Skripty** – test zapsán jako skript v některém ze skriptovacích jazyků, které TestComplete podporuje.

Oba typy testů je možno jak nahrávat, tak i zapisovat, v případě keyword testů spíše naklikávat, manuálně. Ovšem nahrávání delších sekvencí testů vede k tomu, že se nám v testech opakují ty samé úkony na několika místech a dlouhodobá údržba takovýchto testů je velmi obtížná.

6.3.1.1 Keyword tests

Keyword testy jsou definované jako stromové struktury, viz obrázek níže. Keyword testy nahrané pomocí funkce Record and Play jsou poměrně přehledné ve srovnání s nahranými skripty. Jeden keyword test je jakousi analogií metody ve skriptech, je zde možné používat proměnné, parametry, volat jiné keyword testy, případně volat skripty.

Záznam vykonávání keyword testů je poměrně přehledný. TestComplete zobrazuje porovnání jednotlivých kroků exekuce daného testu s tím, jak tento keyword test vypadal při nahrávání, případně při referenčním běhu.



Obrázek 6.3 – TestComplete Keyword test editor[59]

Na druhou stranu, keyword testy jsou uchovávány ve formátu XML, který je navíc plný odkazů pomocí různých id. Tudíž případně mergování oprav keyword testu od více lidí je výrazně náročnější než v případě skriptů.

6.3.1.2 Skripty

Při založení projektu se TestComplete zeptá, jaký skriptovací jazyk chceme používat, tuto volbu později nelze změnit a v rámci projektu nelze jednotlivé skriptovací jazyky míchat. Testovací nástroj TestComplete 10 podporuje následující skriptovací jazyky:

- JScipt – Microsoft dialekt jazyka ECMAScript, blíží se specifikaci EcmaScript3. Nepodporuje kruhové reference mezi jednotlivými Units.
- VBScript – neřeší výjimky pomocí try catch, runtime obsahuje globální Err objekt, který obsahuje výsledek poslední operace, výsledek každé operace je potřeba kontrolovat.
- DelphiScript
- C++ Script a C# Script – běží na JScript skriptovacím engine, s jazyky C++ a C# nemají vlastně nic společného. TestComplete dokumentace skriptovací jazyky C++ Script a C# Script nedoporučuje používat, pokud se dané skripty nepoužívají k testování tzv. Connected nebo Self-testing aplikací napsaných v jazycích C++ nebo C#[75].

V TestComplete 11 přibyla ještě podpora pro Python a v TestComplete 12 přibyla podpora pro moderní JavaScript, který je založený na skriptovacím engine V8 od Google. Ovšem během psaní této diplomové práce jsem měl přístup pouze k TestComplete 10, tudíž zde uvedené informace se budou vztahovat k této verzi.

Tabulka 6.1 – Skriptovací jazyky v TestComplete

Jazyk	Reference jako parametry	Kruhové reference	Zpracování chyb
JavaScript	Ne	Ano	try catch
JScript	Ne	Ne	try catch
Python	Ne	Ano	try except
VBScript	Ano	Ano	globalní Err proměnná
DephiScript	Ano	Ano	try except
C# Script	Ne	Ne	try catch
C++ Script	Ne	Ne	try catch

Osobně bych jako skriptovací jazyk doporučil JScript, v TestComplete 12 a v novějších verzích JavaScript, protože JavaScript je velmi rozšířený jazyk, existuje do něj spousta knihoven, které jsou připravené pro běh na různých prostředích s různou mírou podpory jednotlivých vlastností tohoto jazyka. Zároveň je možné v jazyku JavaScript používat různé polyfills pro doplnění různých vlastností jazyka.

JavaScript v TestComplete 10 nepodporuje serializaci/deserializaci do/z JSON, ale tato vlastnost jde jednoduše doplnit pomocí polyfill knihovny[76]. V novějších verzích programu TestComplete už podpora pro serializaci/deserializaci do/z JSON má být zahrnuta[86].

Výhodou skriptovacích jazyků je, že TestComplete API je ve všech skriptovacích jazycích velmi podobné. V dokumentaci aplikace TestComplete je možné přepínat mezi jednotlivými jazyky a všechny funkce jsou zde podrobně popsány. Navíc TestComplete poskytuje některé pomocné objekty, které se snaží poskytovat stejnou úroveň funkčnosti ve všech skriptovacích jazycích:

- **`aqConvert`** – konverze mezi datovými typy
- **`aqEnvironment`** – poskytuje informace o operačním systému, dostupných TestComplete plugins, atd.
- **`aqObject`** – práce s objekty za běhu, ukládání TestComplete objektů.
- **`aqUtils`** – nabízí různé pomocné metody.
- **`BuiltIn`**, **`Utilities`** a **`Win32API`** – zastaralé, dostupné pouze z důvodu zpětné kompatibility.

TestComplete jména ovládacích prvků testované aplikace mapuje v tvz. name mapping souboru, jde o XML soubor, který je opět plný různých id a odkazů a je velmi obtížné s ním pracovat ve verzovacím systému. Pokud více testerů pracuje na stejném projektu, tak sloučení nějaké drobné změny dost často vyžaduje opětovné ruční mapování použitých ovládacích prvků, případně je možné name mapping zamergovat v TestComplete, ale tato možnost produkuje mnoho duplicit, čímž dále ztěžuje práci s name mapping souborem.

TestComplete umožňuje testy nahrát, tj. zapnout nahrávání a pak manuálně provádět testovací scénář. Pokud jsou takto nahrávány delší testy, tak jsou poměrně nepřehledné. Navíc jsou

nahrávané testy provázané právě s name mapping souborem. Pokud je potřeba tyto testy parametrizovat, nebo je později změnit, pak i jednoduché změny zabírají mnoho času. Z osobní zkušenosti bych nahrávání testů doporučil, jen k získání nějakého testu, se kterým už nebudeme v budoucnosti provádět téměř žádné změny. A pokud očekáváme, že budeme testování na daném projektu provádět bez slučování různých verzí testů od několika lidí.

TestComplete je komerční software, jeho cena se pohybuje přibližně 7 463 € - 8 025 € za TestComplete platform za floating licenci, která není vázaná na konkrétní počítač. Cena licence vázané na konkrétní počítač se pohybuje kolem 3 730 € za licenci. K samotné platformě je ovšem třeba ještě připočíst cenu modulů, která je 2 135 € za floating licenci a 1 067 € za každou node-locked licenci pro každý modul, v případě desktop a web modulů. Nástroj TestExecute, který umožňuje pouhé pouštění testů, je levnější a stojí přibližně 539 - 653 €, tento nástroj má automaticky floating licenci. Platba za TestComplete je jednorázová, ovšem za updates je nutné platit.

6.3.2 Selenium

Selenium je standard pro testovaní web aplikaci. Jde o poměrně starý a odladěný nástroj, který je dostupný zdarma pod Apache 2.0 licencí.

Starší verze se jmenuje Selenium Remote Control. Tato nyní už deprecated verze, stojí na myšlence „proxy injection“, tj. Selenium otevře internetový prohlížeč s proxy nastavenou na lokální URL, kde naslouchá Selenium server. Komunikace pak probíhá přes tuto proxy a do otevřených webových stránek je přidán kód v jazyku JavaScript, který zajišťuje komunikaci se Selenium serverem.

Aktuální verze je Selenium WebDriver. Tato verze používá protokol založený na HTTP, pomocí kterého komunikuje s webovým prohlížečem. To přináší lepší kontrolu internetového prohlížeče a spuštění událostí na DOM.

Existuje SeleniumIDE plugin pro webový prohlížeč Firefox, který umí nahrávat jednotlivé kliky a pak je přehrávat (tj. record-and-playback přístup). Tento nástroj se hodí pro rychlé vytváření jednoduchých skriptů a pro prototypování. Ovšem pro robustní testování reálných aplikací je doporučeno používat Selenium přímo z některého programovacího jazyka, kterých Selenium podporuje několik.[79]

Funkčností se Selenium WebDriver vyrovná TestComplete Web modulu. Rozdíl mezi nástrojem Selenium a TestComplete je ve způsobu provádění testů. Selenium na stránce vyvolává události na DOM. Naproti tomu TestComplete opravdu hýbe s kurzorem myši, což se může u některých speciálních případů hodit.

Pokud nám jde pouze o testování webových aplikací, tak pro většinu případů může být Selenium dostatečné, ovšem pokud potřebujeme testovat i desktopové aplikace a chtěli bychom používat pouze jeden nástroj, musíme používat jiný nástroj, nebo můžeme zvolit níže zmíněnou nástavbu Robot framework.

6.3.3 Robot framework

Robot framework sám o sobě vlastně UI testovat neumí. Tento Framework používá knihovny jako:

- **Selenium/Selenium 2** pro testování webových aplikací.
- **AutoIt** pro testování desktop aplikací na systému Windows.

- **Android library** pro testování aplikací určených pro systém Android.

Výše zmíněné knihovny a mnoho dalších pro Robot framework vykonávají jednotlivé testy. Samotný Robot Framework se zaměřuje na akceptační, behavorial a data driven testování. K zápisu používá keyword-driven přístup, jak je vidět na příkladu níže.

Příklad 6.4 – Behaviour driven test v Robot Frameworku[80]

```
*** Test Cases ***
Add two numbers
    Given I have Calculator open
    When I add 2 and 40
    Then result should be 42

Add negative numbers
    Given I have Calculator open
    When I add 1 and -2
    Then result should be -1

*** Keywords ***
I have ${program} open
    Start Program    ${program}

I add ${number 1} and ${number 2}
    Input Number    ${number 1}
    Push Button     +
    Input Number    ${number 2}
    Push Button     =

Result should be ${expected}
    ${result} =    Get Result
    Should Be Equal    ${result}    ${expected}
```

6.4 Testovatelnost JS aplikací

Aplikace, kterou plánujeme automatizovaně testovat, by měla být na automatizované testování připravená. Generování id různých elementů na stránce by mělo být předpovídatelné, aby bylo možné jednotlivé ovládací prvky pomocí těchto id snadno určit.

Velmi těžko se automatizuje práce s ovládacími prvky na webové stránce, které z pohledu testovací knihovny nemají žádnou vnitřní strukturu. Příkladem takového elementu může být HTML5 Canvas. Například knihovna OpenLayers vykresluje mapu a všechny její vrstvy právě do Canvas elementu, pokud bychom potřebovali automatizovat práci s mapou v Canvas elementu, tak je to velmi obtížné. Pro testovací nástroje jde o neustále se měnící bitmapový obrázek. Ovšem pokud přidáme na webovou stránku nějaký prvek, který bude ukazovat aktuální polohu kurzoru myši nad mapou, tj. který bude ukazovat zeměpisnou šířku a délku, tak to umožní testovat operace pro práci s mapou. Aplikace může tyto pomocné funkce nabízet i skrytě a pouze pro účely automatizovaného testování.

6.5 CompleteTS

V této kapitole představím vlastní řešení na použití skriptovacího API testovacího nástroje TestComplete z jazyka TypeScript. Pomocí tohoto řešení bych chtěl usnadnit práci na testovacích skriptech a zajistit snazší refaktorování těchto skriptů.

Během práce s TestComplete ve verzi 10 jsem narazil na mnohé problémy. Největší problémy jsou s komponentou name mapping, která mapuje ovládací prvky na webových stránkách a v desktopových aplikacích na kratší jména, která jsou používána v keyword testech a ve skriptech. Name mapping je řešen jako jeden velký xml soubor, který může u větších projektů mít velikost až stovky MB. Pokud na projektu pracují dva a více lidí a posílají svoje změny do verzovacího systému, tak v případě konfliktu je merge s pomocí verzovacího systému téměř nemožný. Konflikt nastane vždy, když je do name mapping souboru přidán nový prvek, nebo změněn existující prvek. TestComplete sice nabízí možnost provést merge přímo v aplikaci, ale toto mergování produkuje obrovské množství duplicitních záznamů, čímž se name mapping soubor stává ještě větší a ještě nepřehlednější.

Dalším problémem je obtížné dělání změn v již hotových testech. Refaktoring v již hotových testech je také poměrně obtížný. Pokud se projekt skládá z větší sady testů, která je složena z mixu z keyword testů a scriptů, je velmi těžké dělat jakékoli změny a přejmenování. Pokud například voláme v projektu na mnoha místech nějakou funkci a potřebujeme ji přejmenovat, tak není v TestComplete 10 možné tuto funkci jednoduše přejmenovat, ale musíme vyhledávat jednotlivá použití této funkce. Navíc vyhledávání funguje spolehlivě pouze ve skriptech, v keyword testech se v TestComplete vyhledává velmi obtížně.

Vzhledem k tomu, že TypeScript přináší do jazyka JavaScript pevně danou strukturu a snazší refactoring, tak mě napadlo TypeScript a TestComplete propojit. Právě z toho vyšel projekt **CompleteTS**, který umožňuje používat TypeScript k psaní TestComplete testů. Díky tomu, že při použití CompleteTS je možné mít všechny testy v TypeScript souborech, odpadá problém s obtížným přejmenováním, protože lze použít refaktorovací funkce vývojového prostředí, například ve Visual Studio Code. Name mapping je lepší u větších projektů řešit pomocí hierarchie tříd ve zdrojových souborech v jazyce TypeScript. Name mapping testované aplikace bude rozdělen do více malých pro programátory čitelných TypeScript souborů, jejichž různé verze je možné slučovat ve verzovacích nástrojích stejně snadno jako kterékoli jiné zdrojové kódy. Před spuštěním testů v nástroji TestComplete stačí projekt v jazyce TypeScript zkompilovat a buildovací script nahraje výsledný JavaScript soubor do předem připraveného skriptu v TestComplete projektu. Díky transparentní kompilaci jazyka TypeScript do jazyka JavaScript není problém určit místo v TypeScript projektu podle případných chyb zobrazených ve výsledném JavaScript souboru.

Dále je možné připojit polyfill knihovny, které doplní funkcionalitu, která mi jako programátorovi při práci s TestComplete chyběla. Přidal jsem do projektu knihovnu JSON2 pro serializaci do JSON a tím byla tato funkcionalita snadno doplněna.

6.6 Porovnání testovacích frameworku

V této sekci porovnám jednotlivé testovací frameworky z pohledu vymezeného typu webové aplikace a z pohledu softwarového oddělení, kde jsem moji metodiku ověřoval.

6.6.1 Porovnání nástrojů pro unit testování

V následující tabulce jsem porovnal vybrané testovací frameworky, jsou zde uvedené jejich klady a zápory, licence a to, zda mají testy v daném frameworku přístup k DOM API.

Tabulka 6.2 – Porovnání nástrojů pro unit testovaní

Testovácí nástroje	DOM API	Licence	Klady	Zápory
Jasmine	Ano	MIT	• Testování na cílové platformě • Přehledná syntaxe, popisuje test case • Asynchronní testování • Lze snadno kombinovat s jinými assertion knihovnami	• Sám o sobě neumožňuje automatizované spouštění testů
Mocha.js	Ne	MIT	• File watching[15] • Jednoduché asynchronní testování • Snadno rozšiřitelné	• Pomalejší běh testů
QUnit	Ano	MIT	• Široká podpora • Asynchronní testování • Rychlý běh testů	• Syntaxe není tak přehledná, nepopisuje test case
Karma	Ano	MIT	• File watching[15] • Široká podpora pro mnoho testovacích nástrojů	• Jde pouze o spouštěč testů, závisí na výše uvedených knihovnách

Pro unit testování webových mapových aplikací bych doporučil kombinaci Jasmine s Karma test runner. Jasmine umožňuje testování přímo na cílové platformě, tj. přímo v prohlížeči. Spojení Jasmine s Karma test runner přidá možnost spouštět testy automatizovaně ve více prohlížečích, například v tasku „test“ v build systému Gulp.

6.6.2 Porovnání E2E testovacích frameworků

Pokud jde o testování pouze webových aplikací, tak Selenium, případně Robot framework používající Selenium dostačují.

Problém s použitím Robot frameworku nastává v případě, že je potřeba testovat i složitější desktopové aplikace, což je i případ prostředí, kde pracuji a kde se používají i různé aplikace třetích stran s bohatým GUI. Open-source řešení Robot framework nabízí knihovnu, která používá nástroj AutoIt, který je určený pro ovládání GUI na Windows. Ovšem tento nástroj má problémy s různými typy desktopových GUI frameworků a s pokročilými desktopovými GUI elementy, jako jsou například select-boxes/combo-boxes, tree-view elementy a podobně.

[15] File watching: Sleduje, zda se nějaký soubor ze sledované množiny souborů nezměnil. Pokud ano, tak spustí testy.

Naproti tomu TestComplete zvládá i automatizované testování mnoha těchto pokročilých GUI elementů. I když i TestComplete může mít s některými typy elementů problémy, například automatizace práce s Ribbon panely ve stylu nových MS Office je velmi obtížná.

Přestože TestComplete má u větších projektů problémy s velmi složitým refaktorováním a komponenta name mapping komplikuje spolupráci více lidí na automatizovaných testech v rámci jednoho projektu. Řekl bych, že s použitím hierarchie modulů v jazyce TypeScript a mého vlastního řešení **CompleteTS**, by mělo být možné nalezené problémy odstranit. TestComplete ve spojení s CompleteTS nabízí komplexní řešení pro testování široké škály aplikací od webových, po desktopové a případně i mobilní. Toto řešení významně usnadňuje refaktorování a umožňuje normální používání verzovacích systémů bez nutnosti problematického mergování obrovských name mapping souborů.

Kapitola 7

Metodika vývoje

V této kapitole se budu zabývat metodikou sestavení vhodného vývojového prostředí pro vývoj webových mapových aplikací. Tj. aplikací pro zobrazení map a práci s mapami ve webovém prohlížeči. Představím zde vybranou sadu nástrojů, často označovanou jako tzv. „development stack“, kterou jsem vybral jako vhodnou sadu nástrojů k vývoji mapových webových aplikací. Navrhnuté vývojové prostředí je cílené hlavně na vývojáře back-endu webových aplikací, kteří jsou zvyklí na vývoj v jazycích Java nebo C#.

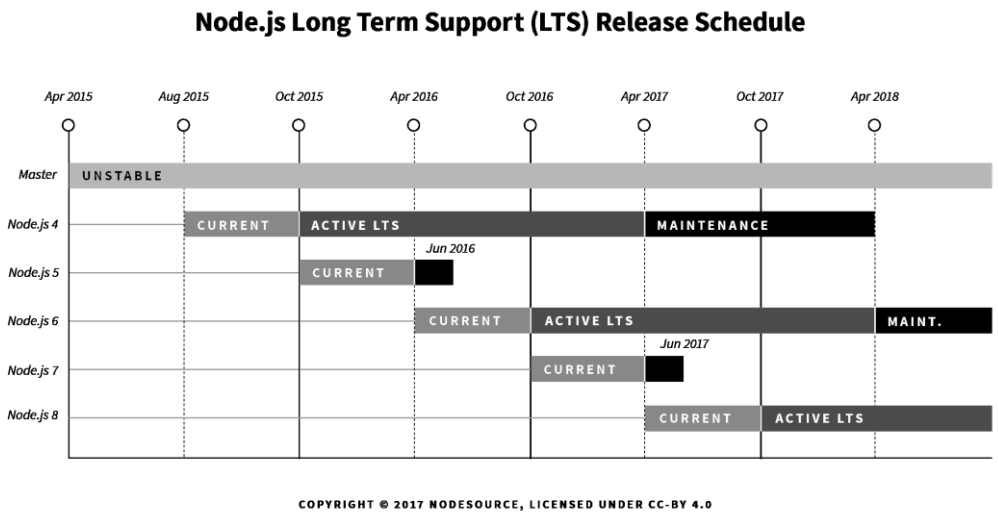
Metodika byla testována na operačním systému Windows, ale vzhledem k tomu, že většina nástrojů je multiplatformní, tak lze tyto postupy nebo velmi podobné postupy aplikovat i na ostatní operační systémy.

7.1 Sestavení vývojového prostředí

Začnu s popisem přípravy prostředí k vývoji, který byl v mém případě prováděn ve vývojovém prostředí Visual Studio Code, ve kterém jsem psal front-endový kód v jazyce TypeScript, který byl při buildu přeložen do jazyka JavaScript.

Jako pomocný nástroj pro psaní back-endového mock serveru jsem zvolil vývojové prostředí Netbeans a aplikační server TomEE.

Jako první krok před vývojem bych doporučil nainstalovat node.js z následující webové stánky: https://nodejs.org/en/. Doporučuji LTS (Long-term support) verzi, která sice neobsahuje nejnovější novinky, ale většina chyb by v ní už měla být vyřešena. Jak je vidět na následujícím obrázku, LTS verze je v případě node.js verze, která už má za sebou několik měsíců používání jako Current verze, a navíc ji v budoucnosti čeká prodloužená podpora.



Obrázek 7.1 - Node.js podpora verzí[43]

7.1.1 Výběr programovacího jazyka

Z představených alternativ k jazyku JavaScript (dále jen JS) jsem zvolil TypeScript (dále jen TS), protože přináší do jazyka JS statické typy, snazší refaktoring ve srovnání s JS a možnost snadno a přehledně členit kód do modulů. Dále TS přináší interface, možnost používat třídy a klasickou třídní dědičnost i ve starších prohlížečích.

V dnešní době (2017) jde o poměrně zavedený programovací jazyk, pro který jsou dostupné kvalitní nástroje, například vývojové prostředí Visual Studio Code. Díky repositáři Definitely typed[25] je dostupná typová kontrola i při volání JS knihoven, které v TS napsané nejsou.

Navíc je tento programovací jazyk používaný softwarovými giganty jako Microsoft a Google, tudíž podpora TS bude zajištěna i v dlouhodobém časovém horizontu.

7.1.2 Instalace kompilátoru jazyka TypeScript

Po nainstalování node.js můžeme přejít k dalšímu kroku, nainstalování TypeScript kompilátoru. K tomu nám stačí otevřít příkazový řádek a tam napsat:

```
npm install -g typescript
```

Tento příkaz nainstaluje aktuální stabilní verzi TypeScript kompilátoru.

7.1.3 Vývojové prostředí

Dále je nutné nainstalovat nějaké vývojové prostředí. TypeScript je možné editovat v mnoha editorech. Já jsem zvolil vývojové prostředí **Visual Studio Code**[16] (dále VS Code). Jde o multiplatformní vývojové prostředí, které funguje na Windows, macOs i Linuxu. Visual Studio Code je stejně

[16] https://code.visualstudio.com/

jako TypeScript pod správou společnosti Microsoft, ovšem jak programovací jazyk TypeScript, tak i zdrojové kódy VS Code jsou pod open-source licencí, Apache License 2.0 pro TypeScript a MIT licence pro VS Code.

Pro vývoj, je potřeba mock server, který poskytuje REST, nebo REST-like API. Toto API poskytuje data a přímá data zadaná uživatelem. Mapová aplikace, na které jsem pracoval, se má připojit na back-end, který běží na JavaEE serveru. Proto jsem pro tvorbu testovacího back-endu zvolil programovací jazyk Java a NetBeans IDE[17]. Ovšem pro vývoj back-endu, případně mock back-endu v Javě je možné použít libovolné Java IDE, např. Eclipse nebo komerční IntelliJ IDEA, v případě, že by uživatel chtěl webovou aplikaci napojit na .Net server, tak lze použít Microsoft Visual Studio.

Pro nasazení mockovacího serveru v JavaEE je možné zvolit například aplikační server TomEE. Zpočátku jsem používal aplikační server Glassfish, ale tento server obsahoval poměrně nepříjemné softwarové chyby, které bylo nutné řešit pomocí různých opravných záplat. Od Ing. Martina Mudry a z mnoha ohlasů na internetu jsem se dozvěděl, že tyto problémy jsou u aplikačního serveru Glassfish poměrně časté. U TomEE bylo použití aplikačního serveru bez potíží.

7.1.4 Výběr build systému

Z build nástrojů představených v této práci bych doporučil Gulp. Přestože je mladší než konkurenční Grunt a má zatím méně doplňků. Gulp je rychlejší a jeho konfigurační soubory se více podobají JavaScriptu a v případě větších projektů jsou přehlednější než konfigurace konkurenčního nástroje Grunt, jehož konfigurace se spíše podobá JSONu. Díky výše popsaným výhodám, lze očekávat, že v budoucnosti bude více používaný právě Gulp, jehož obliba stále roste[93].

7.1.5 Build systém

Pro nainstalování příkazu gulp, zadejte následující příkaz do konzole

```
npm install --global gulp-cli
```

Pak se v konzoli přesuneme do složky projektu a spuštěním následujícího příkazu přidáme Gulp mezi závislosti.

```
npm install --save-dev gulp
```

Ve složce projektu vytvořte soubor gulpfile.js, do kterého umístíte konfiguraci buildu. Výchozí („default“ task) se spouští jednoduše následujícím příkazem, který je nutné spustit opět ve složce daného projektu

```
gulp
```

Je možně spouštět i více tasků, stačí za slovem gulp uvést seznam tasků, co se mají pustit.

7.2 Metodika testování

Pro unit testování doporučuji testovací framework Jasmine, který umožňuje testování přímo na cílové platformě, tj. přímo v prohlížeči. Je možné testovat také ve spojení s Karma test runner, který může Jasmine testy pouštět automatizovaně ve více prohlížečích. Jasmine nabízí velmi přehlednou syntaxi, proto lze jednotlivé testy pohodlně mapovat na test cases, podporuje asynchronní testování a lze ji snadno kombinovat s jinými assertion knihovnami.

[17] https://netbeans.org/

Pro end-2-end testování jsem zvolil TestComplete, protože zároveň provádím i automatizované testování desktopových aplikací a tudíž se mi zdálo vhodné to vše sjednotit na jednom pracovišti pod jedinou testovací platformou.

Robot framework neumí testovat složitější desktopové aplikace. Nástroj AutoIt, na který Robot framework spoléhá při testování desktopových GUI aplikací, neumí ovládat pokročilé desktopové GUI elementy, jako jsou například select-boxes/combo-boxes, tree-view elementy a podobně. Na cílovém prostředí, se kromě webových aplikací používají i desktopové aplikace s bohatým GUI. Proto bych zde nedoporučil použití Robot frameworku.

Ovšem pokud by šlo pouze o testování webových aplikací, tak by Selenium WebDriver, případně Selenium WebDriver i s nástavbou Robot framework, mohl být dostačujícím řešením pro end-2-end testování webových aplikací.

Kapitola 8

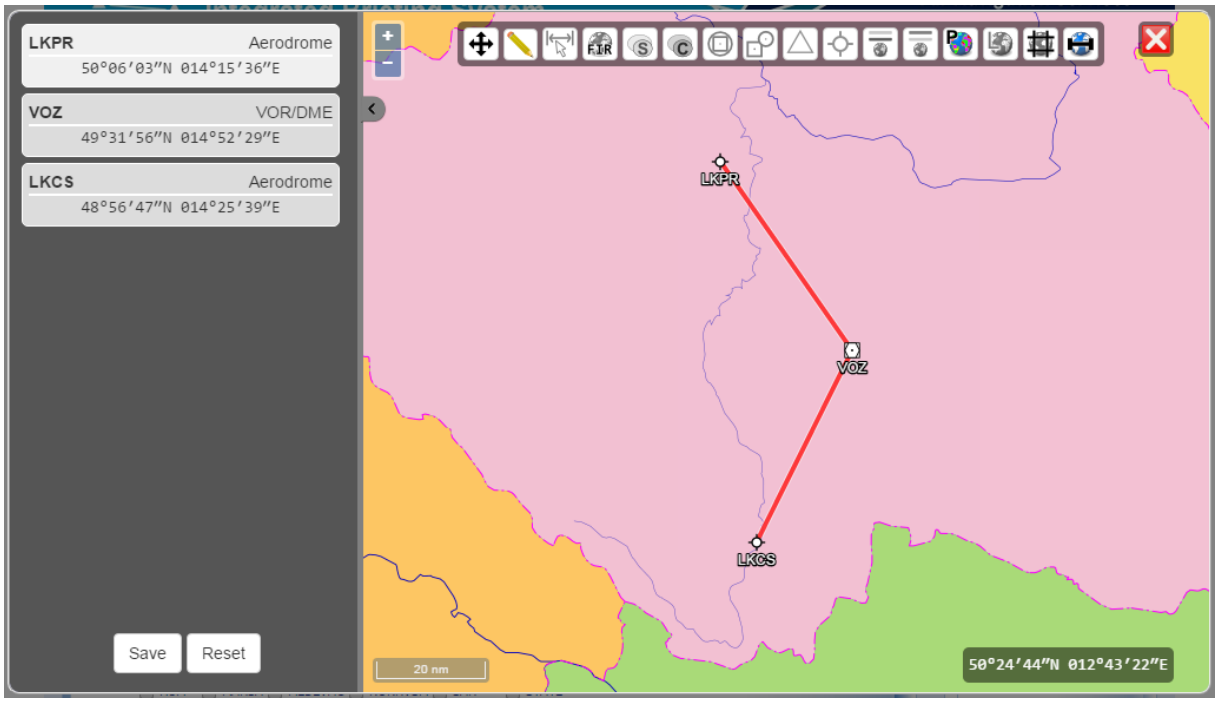
Mapová aplikace v HTML 5

V této kapitole představím aplikaci, na které jsem aplikoval a ověřoval poznatky získané při práci na této diplomové práci.

8.1 Představení aplikace

Jde o webovou aplikaci, která zobrazuje mapu a slouží pro plánování letových tras. Aplikace byla celá napsána v jazyku TypeScript a je postavená na knihovně OpenLayers.

Uživatelské rozhraní je postavené okolo mapy, která je hlavním prvkem aplikace. Vlevo od mapy je zobrazen panel, kde se zobrazují body, které jsou součástí aktuální trajektorie, uživatel zde může přidávat nové body, případně body odstraňovat. Kromě bodů se může trajektorie skládat i z tzv. Airways, což je posloupnost bodu. Uživatel může v levém panelu tyto řetězce rozbalovat a procházet body, ze kterých se skládají. Ve spodní části jsou tlačítka pro uložení mapy na server a tlačítko „Restart“ pro zahození změn provedených v trajektorii.



Obrázek 8.1 – Aplikace pro plánování letových tras

V horním panelu jsou zobrazeny ovládací prvky, které slouží k zapínání a vypínání zobrazení jednotlivých prvků na mapě, zapnutí/vypnutí editace trajektorie, měření vzdálenosti, ke změně pozadí a zobrazení čtvercové souřadnicové sítě, měřítka, atd.

V pravém horním rohu je tlačítko pro zavírání webové mapy a ve spodní části vlevo je zobrazeno měřítko a vpravo jsou zobrazeny souřadnice, na kterých se nachází ukazatel myši.

V případě najetí myší na ovládací prvek se zobrazí pop-up, který popisuje, co daný ovládací prvek dělá.

Trajektorii je možné upravovat, jak přes levý panel, který umožňuje přidávání i odebírání letových bodů, nebo je možné nové body přidávat tahem červené čáry zobrazené mezi existujícími body. Pokud je na daném místě více bodů a není jasné, který z nich chtěl uživatel do letové trajektorie vložit, zobrazí se pop-up okno, kde si uživatel vybere ze seznamu nabízených bodů.

Tabulka 8.1 – Popis funkce jednotlivých tlačítek

Tlačítko	Jméno tlačítka	Popis funkce
	Restartovat přiblížení	Nastaví výchozí přiblížení, tak aby se celá trajektorie vešla na jednu obrazovku
	Editace trasy	Pomocí tohoto tlačítka se zapíná/vypíná funkce editace letecké trasy.
	Měření vzdálenosti	Zapíná/vypíná měření vzdáleností mezi dvěma a více body, vzdálenost je měřena po ortodromě.
	Fir	Zapne/vypne zobrazení FIRů (Flight information regions)
	Speciální vzdušné prostory	Zapne/vypne zobrazení speciálních vzdušných prostorů (Special Use Airspace).
	Kontrolované vzdušné prostory	Zapne/vypne zobrazení kontrolovaných vzdušných prostorů (Controlled Airspace).
	VHF Navaid	Zapne/vypne zobrazení VHF Navaid bodů.
	Non-VHF Navaid	Zapne/vypne zobrazení Non-VHF Navaid bodů.
	Waypoint	Zapne/vypne zobrazení waypointů.
	Letiště	Zapne/vypne zobrazení letišť.
	Nízké letové dráhy	Zapne/vypne zobrazení nízkých letových drah.
	Vysoké letové dráhy	Zapne/vypne zobrazení vysokých letových drah.

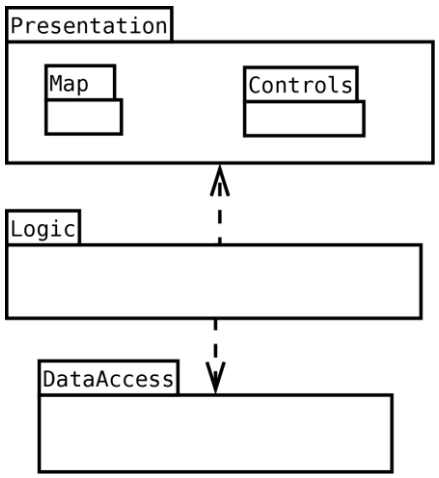
Tabulka 8.1 – Popis funkce jednotlivých tlačítek

Tlačítko	Jméno tlačítka	Popis funkce
	Pozadí	Toto tlačítko umožňuje přepínat mezi jednotlivými pozadími. Ve výchozím stavu je zobrazena politická mapa, další stav je vypnutá mapa na pozadí a poslední stav je šedivá mapa.
	Souřadnicová síť	Zobrazí/skryje souřadnicovou síť. Ve výchozím stavu souřadnicová síť není zobrazena.
	Měřítko	Skryje/zobrazí měřítko v levém dolním rohu. Ve výchozím stavu je měřítko zobrazeno.

8.2 Struktura aplikace

Aplikace se skládá ze třech základních modulů:

- Prezentačního modulu, který se stará o zobrazení trajektorie a ovládacích prvků
- Modulu Logic, který obsahuje objekty, které reprezentují zobrazené body a airways.
- Modulu DataAccess, který zajišťuje komunikaci se serverem přes REST-like API.



Obrázek 8.2 – Moduly Aero aplikace

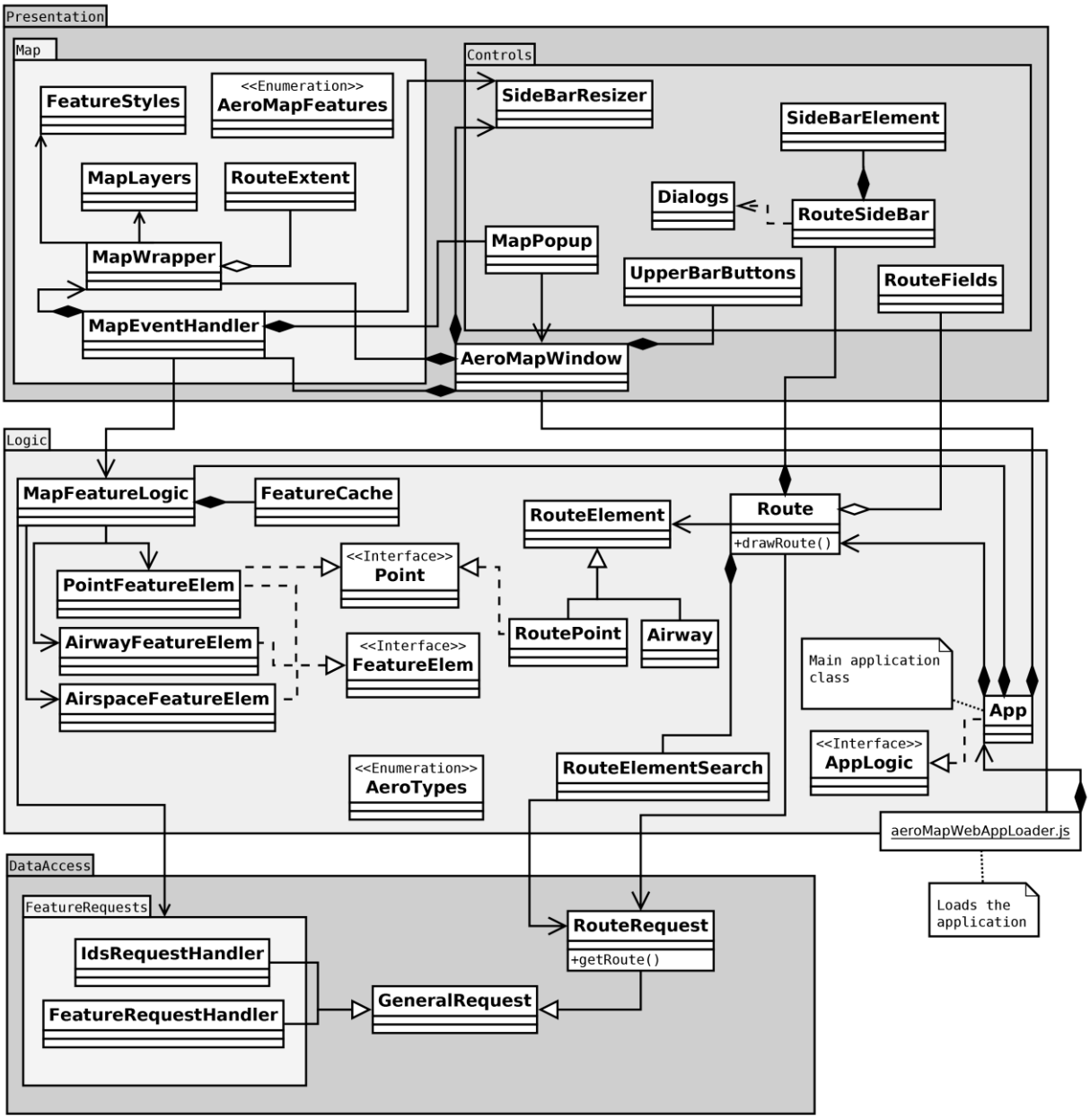
Prezentační modul se dále dělí na:

- Mapový modul, který se zaměřuje na komunikaci s knihovnou OpenLayers, stará se o události, které vzniknou přímo na mapě, a o samotné vykreslování prvků na mapu.
- Controls modul, který se stará o ovládací prvky: tlačítka, pop-up dialogy, postranní panel a události vzniklé v těchto ovládacích prvcích.

V modulu Logic je naznačený aeroMapAppLoader.js, který se stará o načítání zbytku aplikace v případě, že uživatel mapovou aplikaci otevře. Třída App se stará o inicializaci ostatních komponent této aplikace, ať už je inicializuje přímo nebo nepřímo přes nějakou jinou třídu.

Dále se v modulu Logic nachází třída Route, která má na starosti aktuální trajektorii, která je složená z RoutePoints, což jsou jednotlivé body, a Airways, což jsou posloupnosti bodů. FeatureElem je bod, airway nebo airspace, které nejsou součástí letové trajektorie.

V DataAccess modulu je RouteRequest, který se stará o získávání a odesílání letové trajektorie. IdsRequestHandler a FeatureRequest handler se starají o načítání jednotlivých bodů, airways a dalších prvků, které nejsou součástí aktuální letové trajektorie.



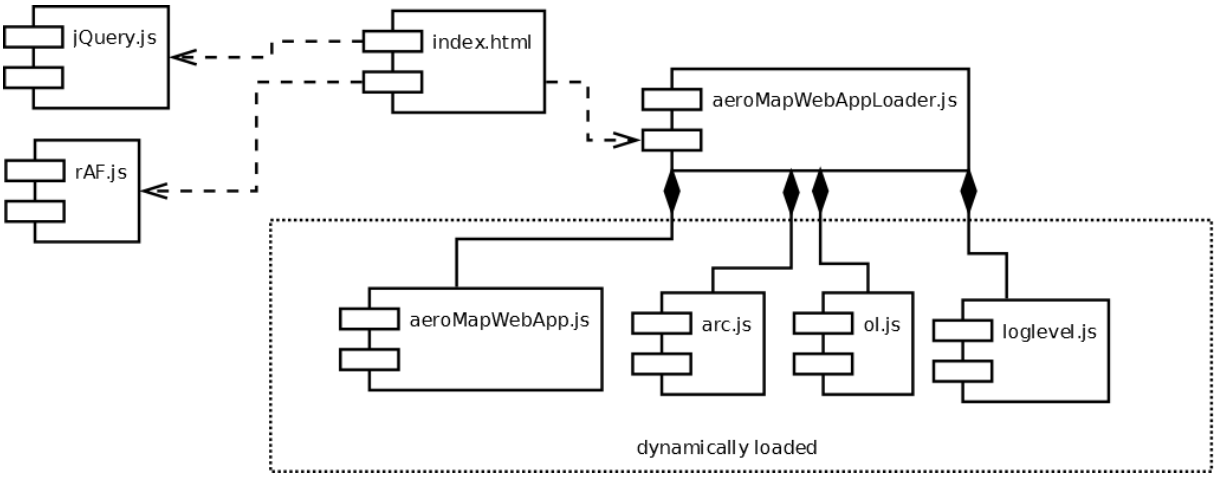
Obrázek 8.3 – Struktura aplikace pro plánování letových tras

8.3 Spouštění aplikace

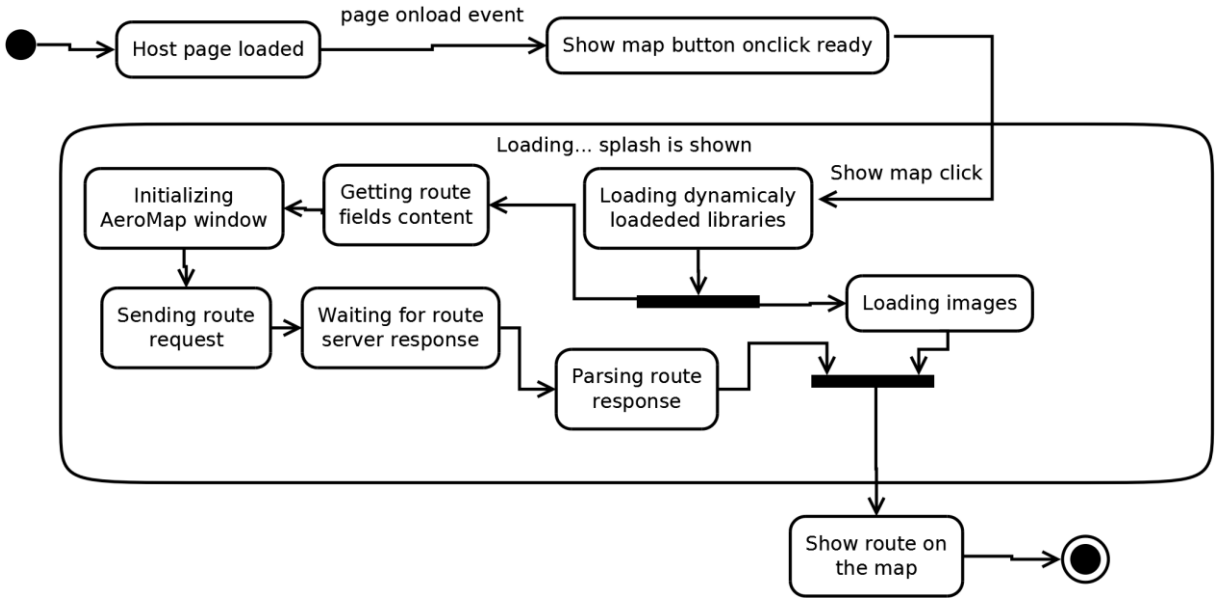
Při zobrazení mateřské stránky, tj. webové stránky, která mapovou aplikaci zapíná, je načtený pouze zavaděč aeroMapWebAppLoader.js. Tento zavaděč obsahuje metodu, která spouští zavádění celé aplikace.

Po vyvolání zobrazení mapy se spustí načítání dynamicky načítaných knihoven jako OpenLayers, arcJs a dalších potřebných knihoven a samotné aplikace pro zobrazení map.

Potom, co jsou všechny knihovny načteny, tak se spustí načítání obrázku, které jsou k provozu aplikace potřeba, a zároveň s tím se ze serveru načtou data výchozí trajektorie. Když jsou všechna data načtena, tak je aplikace zobrazena.



Obrázek 8.4 – Komponenty aplikace



Obrázek 8.5 – Načítání aplikace

Kapitola 9

Závěr

V poslední kapitole shrnu výsledky této práce a navrhnu možná zlepšení do budoucna. Během psaní této práce jsem navrhl metodiku pro vývoj webových mapových aplikací. Navrhnuté technologie a knihovny jsem aplikoval na reálný projekt a vyzkoušel získané poznatky v praxi.

V úvodní části jsem představil webové technologie značkovací jazyk HTML, kaskádové styly a programovací jazyk JavaScript (dále jen JS). To bylo důležité pro další kapitolu, kde jsem hledal vhodnou alternativu k jazyku JavaScript.

V další kapitole jsem z představených alternativ jazyka JavaScript zvolil TypeScript. Podle mého názoru, poměrně vhodný jazyk pro vývoj webových aplikací v případě, že jde o vývojářský tým zvyklý na jazyky jako Java nebo C#. TypeScript činí refactoring snazší v porovnání s jazykem JS, kde může být provádění změn ve větším projektu poměrně obtížné. I podle svých tvůrců je TypeScript cílený na větší projekty, protože se do nich snaží vnést řád známý ze staticky typovaných jazyků. Zaměření na větší a dlouhotrvající projekty naznačuje i moto jazyka TypeScript, které zní „JavaScript that scales“[92].

Dále jsem se zaměřil na vývojové prostředí v jazyce JS, snažil jsem se čtenáři představit pomocné technologie, které se používají při vývoji v jazyce JS. Od běhového prostředí Node.js, ve kterém běží většina nástrojů spojených z vývojem v jazyce JS, po buildovací nástroje, z nichž mě nejvíce zaujal nástroj Gulp.

Potom jsem se zaměřil na JS knihovny pro zobrazení map. Pátral jsem po knihovně, která bude schopná fungovat i v prostředí bez připojení k internetu, například ve firemní síti. Proto jsem zvolil knihovnu OpenLayers, která nabízí ucelenou množinu funkcí a není třeba příliš spoléhat na různé doplňky jako v případě knihovny Leaflets, která je v pokročilých funkcích závislá na doplňcích, které ale s různými verzemi knihovny nemusejí vždy dobře fungovat.

Dále jsem se zaměřil na testování v prostředí jazyka JS. Jako nejlepší řešení se mi zdálo použití testovacího frameworku Jasmine pro unit testování, případně Jasmine ve spojení s Karma test runner.

V případě end-2-end testování, záleží na prostředí, kde se budou dané testy nasazovat. Pokud jde pouze o testování webových aplikací, tak by dostačoval nástroj Selenium. Protože v cílovém prostředí k této webové aplikaci náležejí i další desktopové programy, zvolil bych nástroj TestComplete, který zvládne otestovat téměř každou aplikaci.

V další kapitole jsem stručně popsal výslednou metodiku vývoje a přípravu prostředí pro tuto metodiku.

Na závěr jsem popsal projekt reálné webové mapové aplikace, na které jsem ověřoval poznatky zjištěné v předchozích kapitolách.

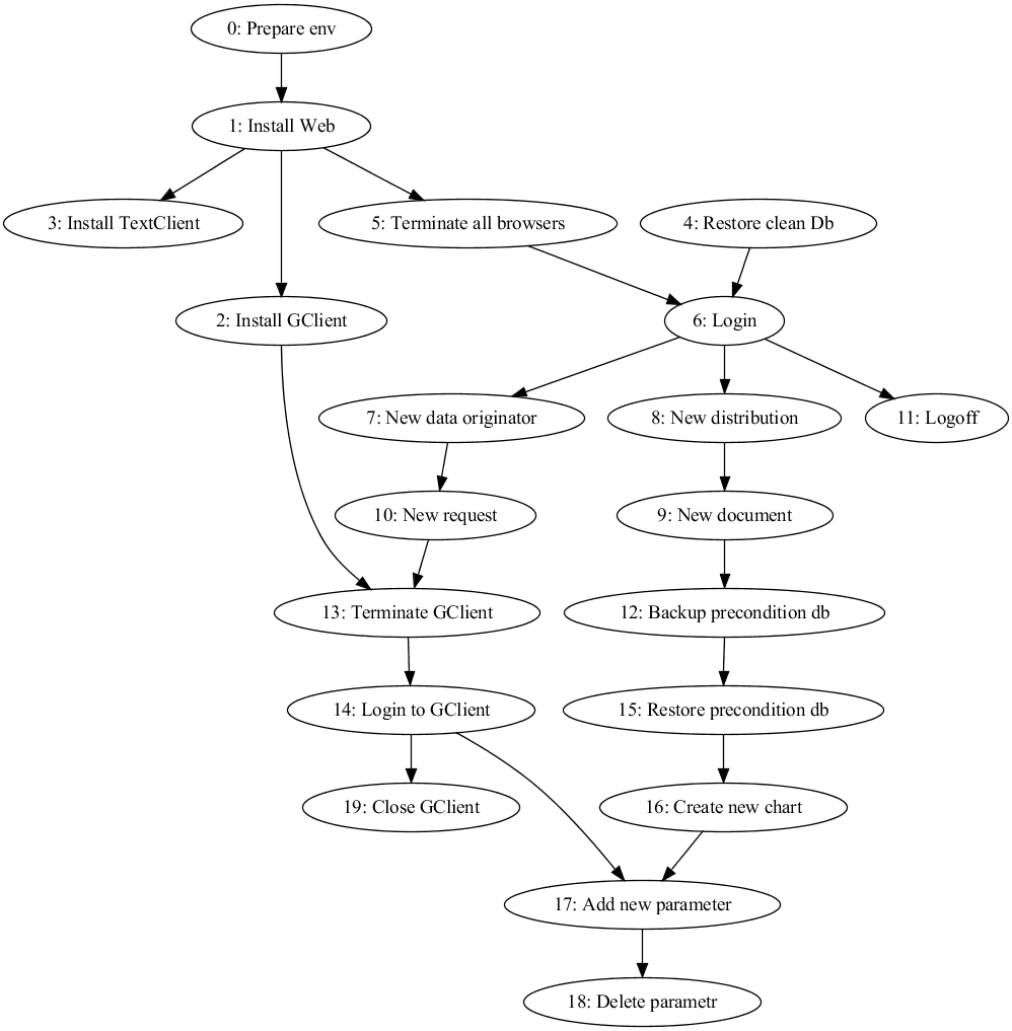
9.1 Možná rozšíření

Po dokončení této práce bych chtěl rozvíjet koncept CompleteTS, který zajišťuje lepší udržitelnost TestComplete testů. Také bych rád propojil CompleteTS s nástrojem DepVis, který slouží k vizualizaci vztahů mezi jednotlivými test cases. Dále bych rád nástroj DepVis používal pro zobrazení výsledků testů v CompleteTS.

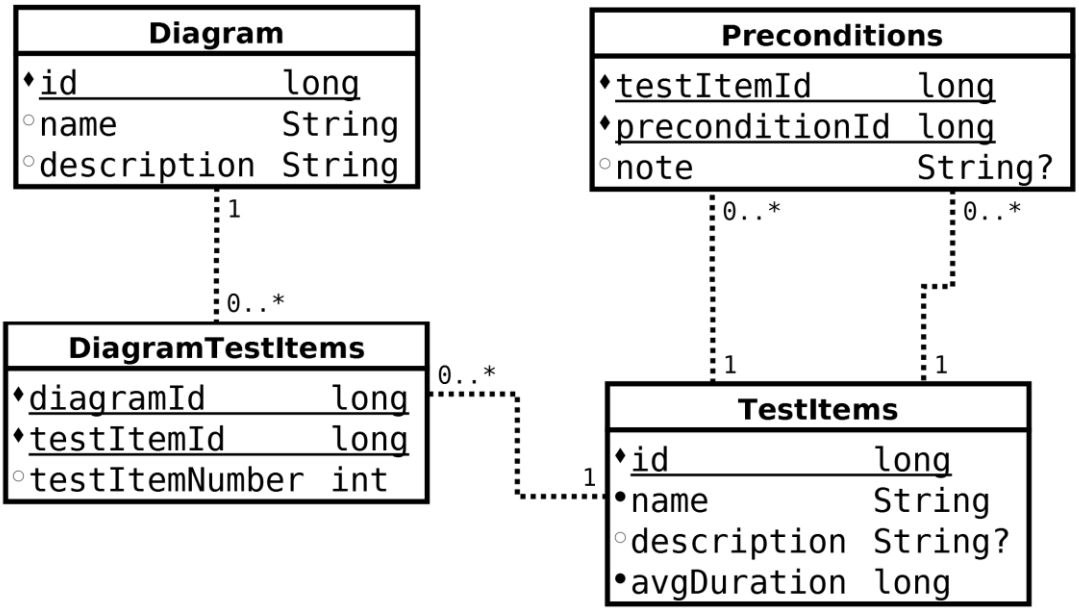
9.1.1 DepVis

DepVis je nástroj, který momentálně vyvíjím. Slouží ke správě vztahů mezi test cases a jejich vizualizaci pomocí open-source software Graphviz. Back-end nástroje DepVis je postavený na platformě JavaEE a front-end je napsaný v jazyce JavaScript s použitím frameworku AngularJS.

Nástroj DepVis bude umožňovat zobrazení jednotlivých test cases v tabulkové formě. Navíc bude zobrazovat seznam prerekvizit každého test case a výpis test case, u kterých je daný test case prerekvizitou. Uživatel bude moci přidávat, editovat a odebírat jednotlivé test cases a vztahy mezi nimi.



Obrázek 9.1 – Ukázka výstupu aplikace DepVis

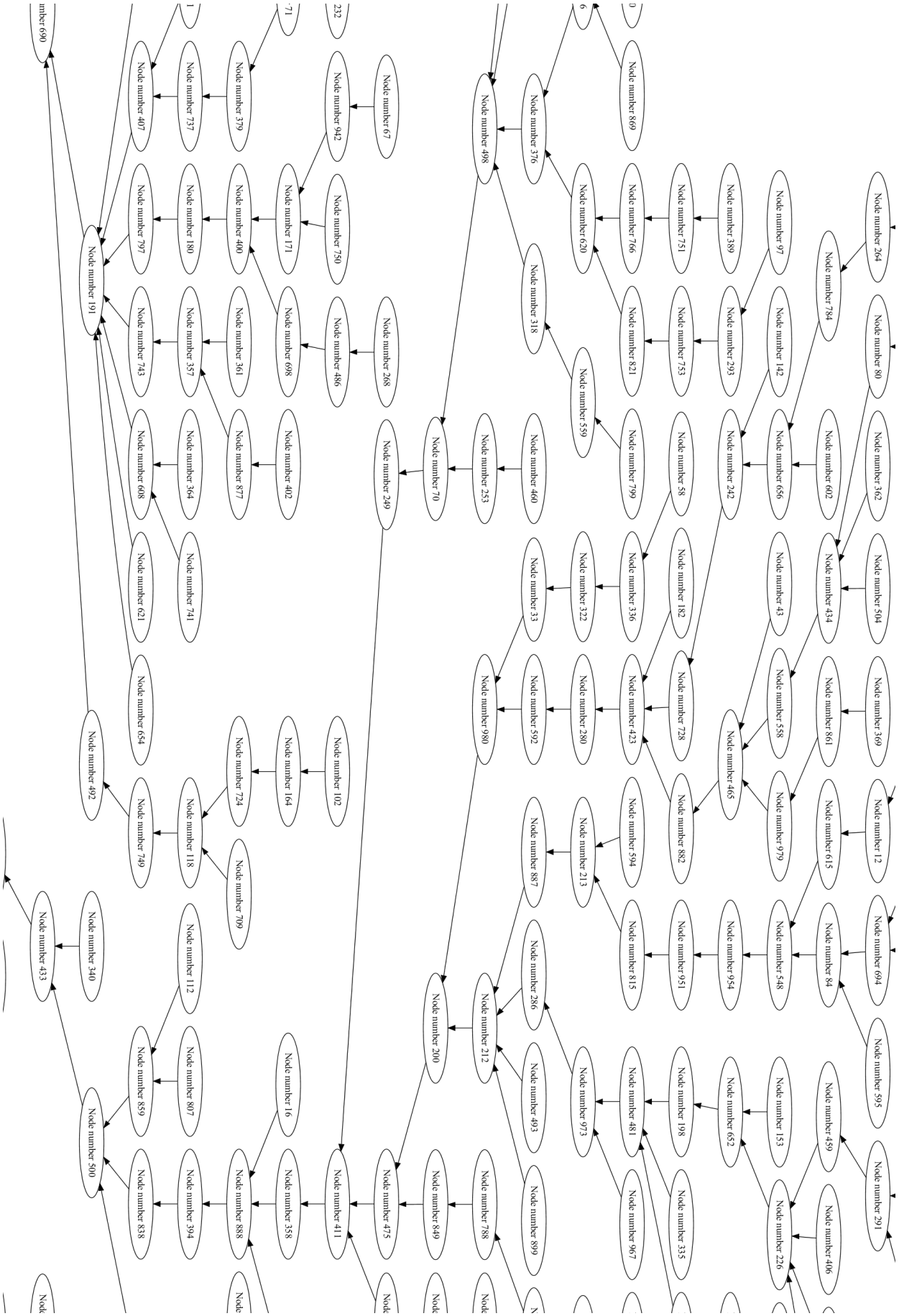


Obrázek 9.2 – Struktura databáze aplikace DepVis

V dalších verzích bych chtěl přidat možnost zobrazení vizualizace výpisu běhů testů do těchto grafů, kde úspěšný test case by se zobrazoval zeleně, neúspěšný červeně a test case, který nemohl být proveden z důvodu selhání prerekvizity by byl vyplněn šedou barvou. Tím by uživatelé získaly rychlý přehled o stavu testované aplikace a závislostech problémových test cases. Testovací software by měl informace o vztazích mezi test cases čerpat přes REST, případně REST-like API, aby se při testování zbytečně neztrácel čas s test case, kde selhaly prerekvizity.

Vizualizační software Graphviz se ukázal jako nejlepší řešení, dokázal v přehledné formě zobrazit i tisíc test cases, což je podle mě maximum, kde vizualizace vztahů ještě dává smysl. Viz obrázek níže. V dalších verzích bych chtěl přidat i možnost selektivní vizualizace částí grafu s problémovými uzly a jejich nejbližšími sousedy.

S nástrojem Graphviz jsem pěkných grafů dosáhl velmi rychle přibližně po třech hodinách práce včetně přípravy vývojového prostředí v JavaEE a propojením programu s vizualizačním nástrojem Graphviz. Proto ho budu nejspíše používat i při dalším vývoji.



Obrázek 9.3 – DepVis – Výřez grafu s tisícem uzlů

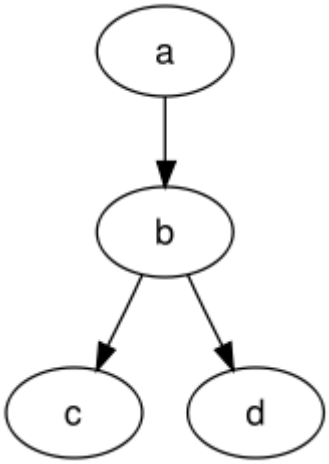
Kromě nástroje Graphviz jsem ještě testoval vizualizační software JUNG. Na rozdíl od rychlé práce s nástrojem Graphviz se mi s JUNG nepodařilo v krátkém čase (odhadem čtyři hodiny) získat grafy v očekávané formě, tj. s automatickým rozvrhováním, přehledné a s minimem křížení vazeb mezi jednotlivými uzly.

9.1.2 Skriptovací jazyk DOT

Práce s nástrojem Graphviz je velmi snadná. Graphviz používá skriptovací jazyk DOT[90], který je velmi intuitivní, viz následující skript a obrázek vygenerovaný tímto skriptem.

Příklad 9.1 – Skriptovací jazyk DOT[91]

```
digraph jmenografu {
     a -> b -> c;
     b -> d;
}
```



Obrázek 9.4 - Obrázek generovaný pomocí výše uvedeného DOT skriptu[91]

Literatura

[1] Flanagan, David. JavaScript: the definitive guide, 6th Edition. " O'Reilly Media, Inc.", 2006.

[2] Fenton, Steve. TypeScript Succinctly. Syncfusion. 2013

[3] Meszaros, Gerard (2007). xUnit Test Patterns: Refactoring Test Code. Addison-Wesley. ISBN 978-0-13-149505-0.

[4] HTML 5 differences from HTML 4: https://www.w3.org/TR/2009/WD-html5-diff-20090423/, citováno 7. 3. 2017

[5] The birth of the web: https://home.cern/topics/birth-web, citováno 7. 3. 2017

[6] HTML: https://en.wikipedia.org/wiki/HTML, citováno 10. 2. 2017

[7] Cascading Style Sheets: https://en.wikipedia.org/wiki/Cascading_Style_Sheets, citováno 10. 2. 2017

[8] JavaScript Guide: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide, citováno 10. 2. 2017

[9] JavaScript: https://en.wikipedia.org/wiki/JavaScript, citováno 10. 2. 2017

[10] JScript: https://en.wikipedia.org/wiki/JScript, citováno 10. 2. 2017

[11] JavaScript: https://cs.wikipedia.org/wiki/JavaScript, citováno 7. 3. 2017

[12] JavaScript for C# Developers: 6. Closures: http://www.charlesnurse.com/Blog/Post/1581/JavaScript-for-C-Developers-6-Closures, citováno 3. 3. 2017

[13] JavaScript Closures: https://www.w3schools.com/js/js_function_closures.asp, citováno 7. 3. 2017

[14] http://javascriptissexy.com/understand-javascripts-this-with-clarity-and-master-it/, citováno 7. 3. 2017

[15] Crockford, Douglas. JavaScript: The World's Most Misunderstood Programming Language http://www.crockford.com/javascript/javascript.html, citováno 10. 2. 2017

[16] Web Workers API: https://developer.mozilla.org/en-US/docs/Web/API/Web_Workers_API, citováno 7. 3. 2017

[17] https://cordova.apache.org/, citováno 10. 3. 2017

[18] https://en.wikipedia.org/wiki/Apache_Cordova, citováno 10. 3. 2017

[19] http://kangax.github.io/compat-table/es5/, citováno 10. 3. 2017

[20] https://www.w3.org/wiki/Graceful_degradation_versus_progressive_enhancement, citováno 14. 3. 2017

[21] https://github.com/jashkenas/coffeescript/wiki/List-of-languages-that-compile-to-JS, citováno 14. 3. 2017

[22] http://www.typescriptlang.org/docs/tutorial.html, citováno 17. 3. 2017

[23] https://en.wikipedia.org/wiki/TypeScript, citováno 17. 3. 2017

[24] http://definitelytyped.org/, citováno 17. 3. 2017

[25] https://github.com/DefinitelyTyped/DefinitelyTyped, citováno 17. 3. 2017

[26] https://en.wikipedia.org/wiki/CoffeeScript, citováno 4. 4. 2017

[27] http://coffeescript.org/, citováno 4. 4. 2017

[28] http://jameslavin.com/Little_Book_on_CoffeeScript.pdf, citováno 4. 4. 2017

[29] A Tour of the Dart Language, https://www.dartlang.org/guides/language/language-tour, citováno 4. 4. 2017

[30] Dart (programming language), https://en.wikipedia.org/wiki/Dart_(programming_language), citováno 4. 4. 2017

[31] https://www.dartlang.org/samples, citováno 4. 4. 2017

[32] Closure Compiler, https://developers.google.com/closure/compiler/, citováno 4. 4. 2017

[33] https://en.wikipedia.org/wiki/Node.js, citováno 4. 4. 2017

[34] https://en.wikipedia.org/wiki/Tiled_web_map, citováno 5. 4. 2017

[35] http://wiki.openstreetmap.org/wiki/Slippy_map_tilenames, citováno 5. 4. 2017

[36] https://en.wikipedia.org/wiki/Web_Mercator, citováno 5. 4. 2017

[37] https://github.com/lukehoban/es6features#enhanced-object-literals, citováno 10. 4. 2017

[38] https://github.com/Microsoft/TypeScript/wiki/TypeScript-Editor-Support, citováno 10. 4. 2017

[39] https://code.visualstudio.com/, citováno 10. 4. 2017

[40] https://github.com/Microsoft/vscode-chrome-debug, citováno 11. 4. 2017

[41] https://palantir.github.io/tslint/, citováno 11. 4. 2017

[42] https://code.visualstudio.com/docs/editor/versioncontrol, citováno 11. 4. 2017

[43] https://github.com/nodejs/LTS#lts-schedule1, citováno 11. 4. 2017

[44] https://github.com/IvanSanchez/leaflet-vs-openlayers-slides, citováno 11. 4.2017

[45] OpenLayers, https://en.wikipedia.org/wiki/OpenLayers, citováno, 16. 4. 2017

[46] OpenLayers API, https://openlayers.org/en/latest/apidoc/index.html, citováno 16. 4. 2017

[47] Modest maps, https://github.com/stamen/modestmaps-js, citováno 11. 4. 2017

[48] Google trends, srovnání popularity programovacích jazyků TypeScript, Dart a CoffeeScript, https://trends.google.com/trends/explore?cat=5&date=2010-11-03%202017-04-11&q=%2Fm%2F0n50hxv,%2Fm%2F0h52xr1,%2Fm%2F0hjc5m0, citováno 11. 4. 2017

[49] Google Closure – Compilation levels, https://developers.google.com/closure/compiler/docs/compilation_levels, citováno 12. 4. 2017

[50] TSLint VS Code plugin, https://marketplace.visualstudio.com/items?itemName=eg2.tslint, citováno 12. 4. 2017

[51] TestComplete documentation, https://support.smartbear.com/testcomplete/docs/, citováno 12. 4. 2017

[52] Build autmation with vanilla JavaScript, https://medium.com/@tarkus/build-automation-with-vanilla-javascript-74639ec98bad, citováno 16. 4. 2017

[53] https://scotch.io/tutorials/automate-your-tasks-easily-with-gulp-js, citováno 16. 4. 2017

[54] https://scotch.io/tutorials/a-simple-guide-to-getting-started-with-grunt, citováno 16. 4. 2017

[55] JavaScript Test Runner, https://github.com/karma-runner/karma/raw/master/thesis.pdf, citováno 18. 4. 2017.

[56] Unit testing, https://en.wikipedia.org/wiki/Unit_testing, citováno 18. 4. 2017

[57] JS interop, https://webdev.dartlang.org/guides/web-programming#js-interop, citováno 27. 4. 2017

[58] Dart for the Entire Web, http://news.dartlang.org/2015/03/dart-for-entire-web.html, citováno 28. 4. 2017

[59] Angular 4: TypeScript, https://www.ng-book.com/2/p/TypeScript/, citováno 4. 4. 2017

[60] KeywordTesting, https://support.smartbear.com/testcomplete/docs/keyword-testing/overview.html, citováno 3. 5. 2017

[61] JScript, https://en.wikipedia.org/wiki/JScript, citováno 3. 5. 2017

[62] An Introduction to LESS, https://www.smashingmagazine.com/2011/09/an-introduction-to-less-and-comparison-to-sass/, citováno 3. 5. 2017

[63] Less (stylesheet language), https://en.wikipedia.org/wiki/Less_(stylesheet_language), citováno 11. 5. 2017

[64] Earthquakes Heatmap, https://openlayers.org/en/latest/examples/heatmap-earthquakes.html, citováno 4. 5. 2017

[65] Test double, https://en.wikipedia.org/wiki/Test_double, citováno 4. 5. 2017

[66] Learn Less in 10 Minutes, http://tutorialzine.com/2015/07/learn-less-in-10-minutes-or-less/, citováno 5. 5. 2017

[67] Software testing, https://en.wikipedia.org/wiki/Software_testing, citováno 11. 5. 2017

[68] Integration testing, https://en.wikipedia.org/wiki/Integration_testing, citováno 11. 5. 2017

[69] Top-down and bottom-up design, https://en.wikipedia.org/wiki/Top-down_and_bottom-up_design, citováno 11. 5. 2017

[70] Jasmine – introduction.js, https://jasmine.github.io/2.0/introduction.html, citováno 12. 5. 2017

[71] Karma – github page, https://github.com/karma-runner/karma, citováno 12. 5. 2017

[72] Java and Google Chrome Browser, https://www.java.com/en/download/faq/chrome.xml, citováno 15. 5. 2017

[73] NPAPI Plugins in Firefox, https://blog.mozilla.org/futurereleases/2015/10/08/npapi-plugins-in-firefox/, citováno 15. 5. 2017

[74] Google Chrome version history, https://en.wikipedia.org/wiki/Google_Chrome_version_history, citováno 15. 5. 2017

[75] Selecting the Scripting Language, https://support.smartbear.com/testcomplete/docs/scripting/selecting-the-scripting-language.html, citováno 15. 5. 2017

[76] JSON in JavaScript, https://github.com/douglascrockford/JSON-js, citováno 15. 5. 2017

[77] TestComplete - Specifics of Usage Common for All Languages, https://support.smartbear.com/testcomplete/docs/scripting/specifics/common.html, citováno 15. 5. 2017

[78] Selenium Documentation, http://www.seleniumhq.org/docs/index.jsp, citováno 16. 5. 2017

[79] Selenium Introduction, http://www.seleniumhq.org/docs/01_introducing_selenium.jsp, citováno 16. 5. 2017

[80] Robot Framework, User Guide, http://robotframework.org/robotframework/latest/RobotFrameworkUserGuide.html, citováno 16. 5. 2017

[81] QUnit, https://qunitjs.com/, citováno 19. 5. 2017

[82] Google map, simple example, https://google-developers.appspot.com/maps/documentation/javascript/examples/full/map-simple, citováno 19. 5. 2017

[83] Leaflets plugins, http://leafletjs.com/plugins.html, citováno 19. 5. 2017

[84] Modestmaps.js, https://github.com/stamen/modestmaps-js, citováno 19. 5. 2017

[85] Just Say No to More End-to-End Tests, https://testing.googleblog.com/2015/04/just-say-no-to-more-end-to-end-tests.html, citováno 19. 5. 2017

[86] TestComplete, JSON support, https://support.smartbear.com/viewarticle/77297/, citováno 19. 5. 2017

[87] Formáty podporované knihovnou OpenLayers, dokumentace pro ol.format, http://openlayers.org/en/latest/apidoc/ol.format.html, citováno 24. 5. 2017

[88] OpenLayers - ol.proj.Projection, http://openlayers.org/en/latest/apidoc/ol.proj.Projection.html, citováno 24. 5. 2017

[89] Ol-Cesium, http://openlayers.org/ol-cesium/, citováno 24. 5. 2017

[90] Graphviz, https://en.wikipedia.org/wiki/Graphviz, citováno 24. 5. 2017

[91] DOT (graph description language), https://en.wikipedia.org/wiki/DOT_(graph_description_language), citováno 24. 5. 2017

[92] TypeScript, https://www.typescriptlang.org/, citováno 24. 5. 2017

[93] Google Trends: Gulp vs Grunt, https://trends.google.com/trends/explore?q=Gulp,Grunt, citováno 24. 5. 2017

Příloha A

Obsah přiloženého CD

Přílohou této diplomové práce je CD, které obsahuje následující složky:

- Složku `CompleteTS`
  - Zdrojové kódy a další soubory spojené s projektem CompleteTS
- Složku `DepVis`
  - Zdrojové kódy projektu DepVis
- Složku `DepVis_examples`
  - Ukázkové grafy vygenerované pomocí nástroje DepVis
- Složku `Jasmine`
  - Příklad jednoduchého testu ve frameworku Jasmine
- Složku `Thesis`
  - Text této práce ve formátech docx a pdf