Pedagogická fakulta
Faculty of Education

Jihočeská univerzita v Českých Budějovicích
University of South Bohemia in České Budějovice

Jihočeská univerzita v Českých Budějovicích

Pedagogická fakulta

Katedra informatiky

Tvorba webových aplikací jazykem ELM

Creation of web applications by ELM language

Vypracoval: Vladimír Jiroudek

Vedoucí práce: PaedDr. Petr Pexa, Ph.D.

České Budějovice 2018

JIHOČESKÁ UNIVERZITA V ČESKÝCH BUDĚJOVICÍCH
Fakulta pedagogická
Akademický rok: **2016/2017**

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení: **Vladimír JIROŮDEK**
Osobní číslo: **P15665**
Studijní program: **B7507 Specializace v pedagogice**
Studijní obor: **Informační technologie a e-learning**
Název tématu: **Tvorba webových aplikací jazykem ELM**
Zadávající katedra: **Katedra informatiky**

Zásady pro vypracování:

Cílem bakalářské práce je zpracování problematiky využití nového funkcionálního programovacího jazyka ELM při tvorbě webových aplikací. ELM je reaktivní a staticky typovaný jazyk, kompilátor, který transformuje zdrojový kód do klientského JavaScriptu, zachytí okamžitě většinu chyb a poskytne v reálném čase jasnou a srozumitelnou zpětnou vazbu. Teoretická část práce bude zaměřena na samotný jazyk ELM, bude představena jeho syntaxe, funkce a budou popsány přínosy použití jazyka ELM při tvorbě webových uživatelských rozhraní. V praktické části bude v jazyce ELM vytvořena sada vlastních ukázkových webových příkladů, které budou použity k názorné demonstraci možností jazyka ELM, jeho výhod a nevýhod v porovnání s klasickým JavaScriptem.

Rozsah grafických prací: **CD ROM**

Rozsah pracovní zprávy: **40**

Forma zpracování bakalářské práce: **tištěná**

Seznam odborné literatury:

1. **Zdroják - elm. Zdroják.cz [online]. Devel.cz Lab [cit. 2017-04-05]. Dostupné z: https://www.zdrojak.cz/?s=elm&submit=**
2. **oRedInk Tech [online]. Evan Czaplicki [cit. 2017-04-05]. Dostupné z: http://tech.noredink.com/**
3. **Single-Page Web Apps in Elm. Linkedin [online]. Kevin Greene, 2016 [cit. 2017-04-05]. Dostupné z: https://www.linkedin.com/pulse/single-page-web-apps-elm-part-one-getting-started-new-kevin-greene**
4. **Elm-lang [online]. Evan Czaplicki [cit. 2017-04-05]. Dostupné z: http://elm-lang.org/**
5. **Creating Your First Elm App. Auth0 [online]. Kim Maida, 2016 [cit. 2017-04-05]. Dostupné z: https://auth0.com/blog/creating-your-first-elm-app-part-1/**

Vedoucí bakalářské práce: **PaedDr. Petr Pexa, Ph.D.**
Katedra informatiky

Datum zadání bakalářské práce: **24. dubna 2017**

Termín odevzdání bakalářské práce: **30. dubna 2018**

Mgr. Michal Vančura, Ph.D.
děkan

L.S.

doc. PaedDr. Jiří Vaníček, Ph.D.
vedoucí katedry

V Českých Budějovicích dne 24. dubna 2017

Prohlášení

Prohlašuji, že svoji bakalářskou práci jsem vypracoval samostatně pouze s použitím pramenů a literatury uvedených v seznamu citované literatury.

Prohlašuji, že v souladu s § 47b zákona č. 111/1998 Sb. v platném znění souhlasím se zveřejněním své bakalářské práce, a to v nezkrácené podobě elektronickou cestou ve veřejně přístupné části databáze STAG provozované Jihočeskou univerzitou v Českých Budějovicích na jejích internetových stránkách, a to se zachováním mého autorského práva k odevzdanému textu této kvalifikační práce. Souhlasím dále s tím, aby toutéž elektronickou cestou byly v souladu s uvedeným ustanovením zákona č. 111/1998 Sb. zveřejněny posudky školitele a oponentů práce i záznam o průběhu a výsledku obhajoby kvalifikační práce. Rovněž souhlasím s porovnáním textu mé kvalifikační práce s databází kvalifikačních prací Theses.cz provozovanou Národním registrem vysokoškolských kvalifikačních prací a systémem na odhalování plagiátů.

V Českých Budějovicích dne 9. července 2018.

Vladimír Jiroudek

Abstrakt/Anotace

Cílem bakalářské práce je zpracování problematiky využití nového funkcionálního programovacího jazyka ELM při tvorbě webových aplikací. ELM je reaktivní a staticky typovaný jazyk. Kompilátor, který transformuje zdrojový kód do klientského JavaScriptu, zachytí okamžitě většinu chyb a poskytne v reálném čase jasnou a srozumitelnou zpětnou vazbu.

Teoretická část práce bude zaměřena na samotný jazyk ELM, bude představena jeho syntaxe, funkce a budou popsány přínosy použití jazyka ELM při tvorbě webových uživatelských rozhraní.

V praktické části bude v jazyce ELM vytvořena sada vlastních ukázkových webových příkladů, které budou použity k názorné demonstraci možností jazyka ELM, jeho výhod a nevýhod v porovnání s klasickým JavaScriptem.

Klíčová slova

ELM, HTML, CSS, JavaScript, webové aplikace

Abstract/Anotation

The aim of this thesis is to describe using the new functional programming language ELM in the creation of web applications. ELM is a reactive a statically typed language. The compiler, which transform the source code into client JavaScript, captures most of the bugs and provides clear feadback in a real time.

Theoretical part will be aimed to ELM itself. The syntax, functions and the benefits of using ELM in creation of web user interfaces will be described.

In the practical part a set of web applications programed by ELM will be created to illustrate the ELM options, advantages and disadvantages compared to classic JavaScript.

Keywords

ELM, HTML, CSS, JavaScript, web applications

Poděkování

Děkuji panu PaedDr. Petru Pexovi, Ph.D. za veškeré rady a pomoc, které mi poskytl při realizaci této bakalářské práce. Jeho připomínky, rady a nápady mi byly pokaždé ku prospěchu a vždy byl pomoc ochotný poskytnout. Velice si jej za to vážím.

Poděkování patří také Barboře Martinkové za projevenou trpělivost a podporu.

Také děkuji mým rodičům za to, že mi studium VŠ umožnili a podporovali mě v něm.

Obsah

1 Úvod

1.1 Cíle

Cíl této bakalářské práce spočívá v popisu jazyka ELM. V popisu jeho možností, funkcionalit, případně potenciálních chyb, které mohou s použitím jazyka ELM při vývoji webových aplikací nastat.

Část práce bude zaměřena na tvorbu webových aplikací jazykem ELM a její porovnání s tvorbou v jazyce JavaScript, který je v současné době nejpoužívanější.

V praktické části bude naprogramována sada webových aplikací jazykem ELM za použití HTML5 a CSS3 technologií. Aplikace budou zvoleny tak, aby na nich bylo možné demonstrovat různý přístup, možnosti a funkce. Aplikace budou podrobně popsány z hlediska jejich funkce a postupů, které budou použity k jejich vytvoření. Finální verze aplikací bude vložena na webové stránky.

1.2 Metody

V úvodu práce bude rozebrán samotný jazyk ELM. Budu se zabývat jeho specifikacemi, funkcionalitami a přípravou prostředí, abychom jej mohli používat.

Bude popsána syntaxe, instalátory, standartní knihovny datové struktury a další. Bude popsáno proč je třeba použít kompilátor do jazyka JavaScript a budou popsány rozdíly, v porovnání s použitím pouze jazyka JavaScript.

Dále bude vytvořena sada aplikací zaměřující se na různé problémy a různá řešení. Podrobně bude vysvětlen způsob naprogramování všech jednotlivých aplikací a problémů spojených s jejich tvorbou. Hotové aplikace nakonec budou umístěny na webové stránky, které budou obsahovat dokumentaci k aplikacím. Stránky budou responzivní tak, abychom viděli jak probíhá zobrazení na různých typech zařízení.

1.3 Východiska

V současné chvíli probíhá tvorba webových aplikací převážně za použití jazyka JavaScript. Využíván je například k tvorbě her, hodin, kalendářů, animací, interaktivních prvků a dalších. To s sebou samozřejmě přináší různá úskalí a problémy, a proto vznikl jazyk ELM, který klade důraz na jednoduchost použití a kvalitní nástroje.

Pro ELM není v současné chvíli dostatek knih ani v anglickém jazyce natož v českém. V českém jazyce je k dostání pouze pár internetových tutoriálů, nicméně do této doby není sepsána žádná obsáhlejší práce, která by jazyk ELM popisovala a zároveň jej porovnávala s JavaScriptem a zdůrazňovala jeho výhody a nevýhody.

2 Seznámení s jazykem ELM

ELM je funkcionální programovací jazyk, který je kompilován do jazyka JavaScript (je zde možnost spuštění ihned po kompilaci pomocí Node.js, nicméně se nepoužívá a vývojáři jej nedoporučují). ELM klade velký důraz na jednoduchost, snadné použití a kvalitní nástroje.[1] Vytvořen byl vývojářem Evanem Czaplickim, který počítá s tím, že bude možné v budoucnu jazyk ELM kompilovat nejen do JavaScriptu, ale třeba i přímo do strojového kódu. Poptávka by mohla být i po kompilaci do nativních mobilních aplikací, nicméně v současné chvíli si musíme vystačit s kompilací pouze do jazyka JavaScript. Po vytvoření dalších kompilátorů by jazyk ELM mohl, čistě teoreticky, konkurovat například i jazyku JAVA. Kompilátor jazyka ELM je napsán v jazyce Haskell.

Obrázek 1: logo jazyka ELM

V současné době je ELM určen pouze k vytváření aplikací, které jsou založené na webovém prohlížeči. ELM je jazyk staticky typovaný (již v době překladu máme kontrolu nad tím, že výsledná aplikace neobsahuje žádnou typovou chybu).

Elm, na rozdíl od jiných jazyků kompilovaných do JavaScriptu, nepřináší pouze nějaké vylepšení jazyka JavaScript nebo jeho nadstavbu. Přináší čistě funkční jazyk, který je určený pro vytváření front-end webových aplikací.[2] Syntaxe jazyka ELM je zcela odlišná od jazyků, na které bývají vývojáři webových aplikací zvyklí (JavaScript, PHP, Java, atd. . .), jelikož nevychází z jazyka C. ELM

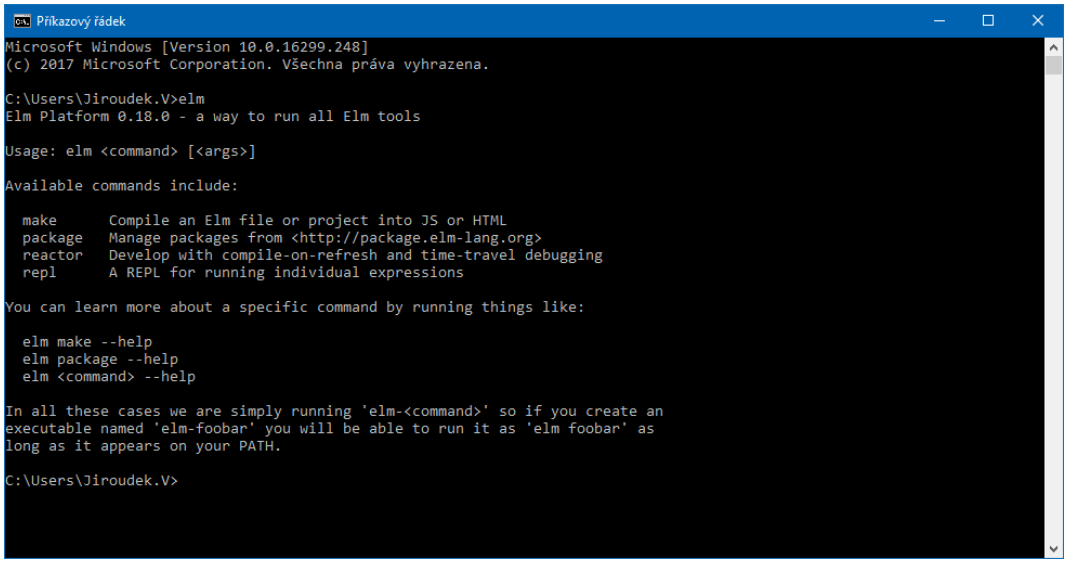
totiž přebírá syntaxi z jazyka Haskell.[3] ELM aplikace je sice kompilována do JavaScriptu, nicméně při programování v ELM nepotřebujeme jazyk JavaScript vůbec znát, jelikož na rozdíl od např. TypeScriptu (jazyk pro vývoj na straně klienta) ELM nepovoluje kombinaci jazyka ELM a jazyka JavaScript.

2.1 Použití ELM

ELM je jazyk, který je v současné chvíli možné použít pouze pro tvorbu webových aplikací. Narozdíl od např. Dartu (poměrně nový jazyk pro vytváření webových aplikací vyvíjený firmou Google, který se ale ukázal být "slepou uličkou"), ELM nepotřebuje pro svůj běh speciální prostředí nebo prohlížeč, jelikož po zkompilování dostáváme čistě kód JavaScript a s ním umí pracovat všechny rozšířené prohlížeče, mobilní nevyjímaje. To nám dává poměrně slušný příslib do budoucna. Použití je doporučováno zejména ve větších projektech, u kterých bude kladen důraz na budoucí údržbu případně nasazování novějších verzí.

2.2 Instalace ELM

Před tím, než začneme pracovat s jazykem ELM a vytvářet pomocí něj webové aplikace musíme ELM nejprve nainstalovat. K tomu slouží instalátor. Dostupný je pro různé počítačové platformy zde. Při psaní této bakalářské verze je to ELM verze 0.18. Další možností, jak ELM nainstalovat je použití NPM (manažer pro instalaci JavaScript balíčků). V tomto případě bychom ELM nenainstalovali globálně, ale pouze lokálně v adresáři projektu. Zde ale bude k demonstraci použito instalace pomocí ELM instalátoru. Po dokončení instalace budete mít k dispozici za pomoci terminálu ve vašem OS několik nových příkazů. Příkaz ELM s různými typy argumentů. Pro zobrazení informace o použití otevřete terminál, napište příkaz „elm“ a zobrazí se Vám přibližně takováto obrazovka:

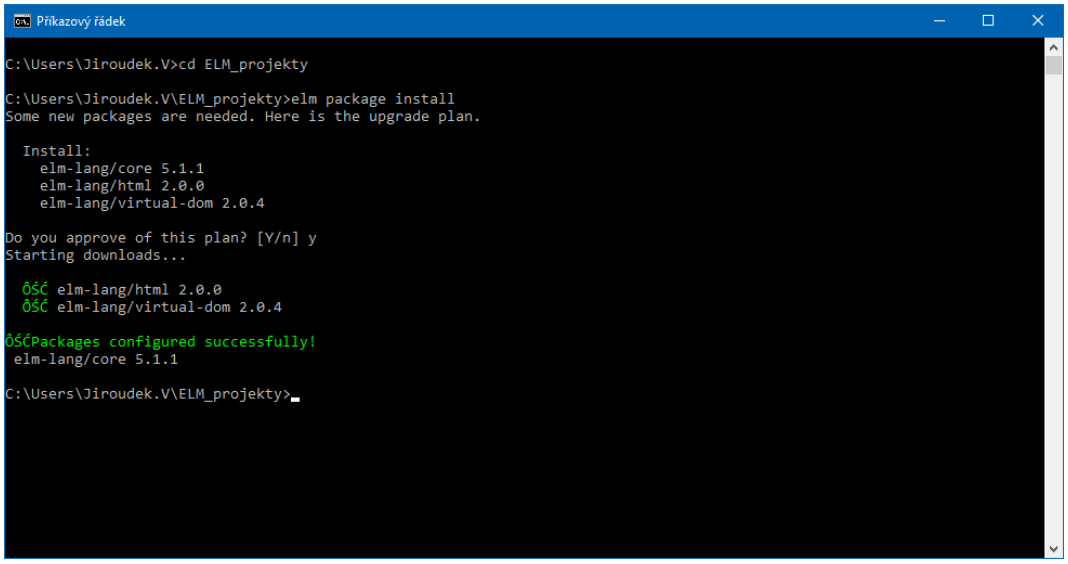


Obrázek 2: Nabídka nástrojů ELM

Jak můžete vidět po instalaci ELM jsme dostali k dispozici čtyři následující nástroje: **elm-repl, elm-reactor, elm-make, elm-package**.

Funkce každého nástroje bude vysvětlena později až jej budeme používat.

Nyní, když máme nainstalován ELM si potřebujeme vytvořit prostor, ve kterém budeme psát samotné aplikace. První co je tedy potřeba udělat je vytvořit adresář pro nový projekt. Jakmile nový adresář vytvoříte přejděte do něj pomocí terminálu. V mém případě jsem se pro daný adresář rozhodl použít označení ELM_projekty. Poté, co přejdeme do daného adresáře využijeme první z dostupných nástrojů a to "elm-package", kterým spustíme instalaci základních balíčků a vytvoření závislostí. Je tedy jasné, že nástroj elm-package slouží k instalaci nových balíčků potřebných k tvorbě nových projektů.



Obrázek 3: elm-package

Došlo k instalaci základních knihoven potřebných pro tvorbu projektů. Tyto knihovny byly nainstalovány do adresáře „elm-stuff“, který se vytvořil v adresáři pro projekt. Dále se nám vytvořil soubor s názvem elm-package.json. Když jej otevřeme v textovém editoru (já používám stejný jako pro tvorbu samotné aplikace – tedy Sublime Text 3) můžeme vidět, jaké základní knihovny máme nainstalované a jejich závislosti.

```
{
    "version": "1.0.0",
    "summary": "helpful summary of your project, less than 80 characters",
    "repository": "https://github.com/user/project.git",
    "license": "BSD3",
    "source-directories": [
        "."
    ],
    "exposed-modules": [],
    "dependencies": {
        "elm-lang/core": "5.1.1 <= v < 6.0.0",
        "elm-lang/html": "2.0.0 <= v < 3.0.0"
    },
    "elm-version": "0.18.0 <= v < 0.19.0"
}
```

Obrázek 4: elm-package.json

Pokud používáte starší verzi ELM musíte ještě doinstalovat balíček „elm-lang / html“, ve verzi 0.18 je však tento balíček nainstalován automaticky. Tímto je základ jazyka ELM nutný pro tvorbu webových aplikací nainstalován.

2.3 ELM nástroje

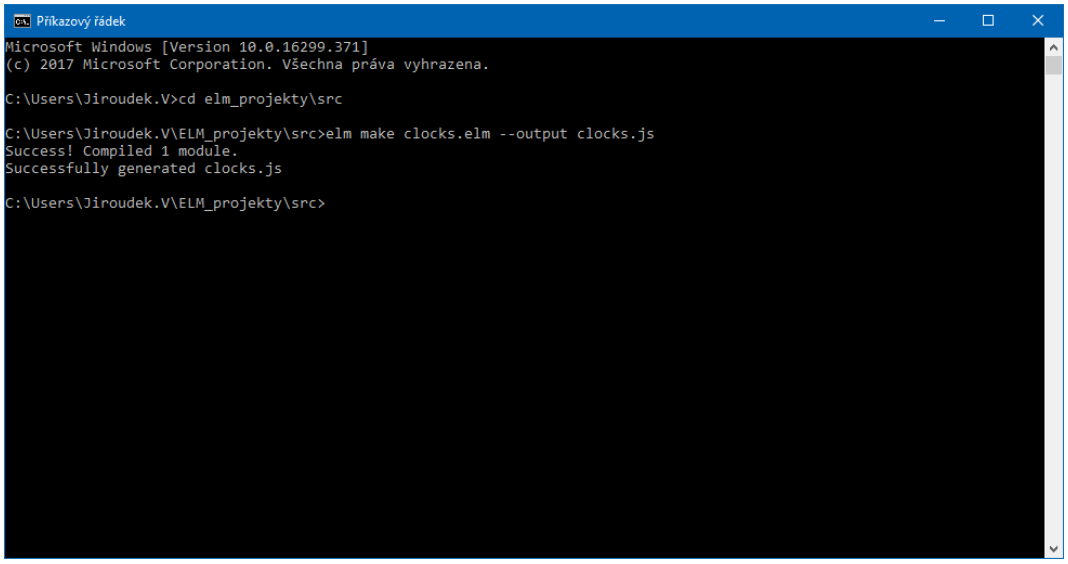
Jak již bylo výše popsáno při instalaci ELM do počítače nám byly poskytnuty 4 následující nástroje, jejichž obsluhu řídíme skrze příkazový řádek v OS: **elm-repl, elm-reactor, elm-make, elm-package**, přičemž elm-package je správce balíčků, elm-repl slouží k vyzkoušení kódu, elm-make je nástroj určený pro kompilaci a elm-reactor nám poskytuje localhost server pro běh ELM aplikací.[4]

2.3.1 Elm-package

Elm-package je nástroj, jež nám slouží ke stahování balíčků potřebných v daném projektu, který vytváříme. Například pokud víme, že budeme potřebovat v projektu, aby jej uživatel mohl ovládat myší (přetahovat položky, zaškrtávat checkboxy atd..) musíme nainstalovat balíček elm-lang/mouse. Uděláme to tak, že pomocí příkazového řádku ve složce, kde máme elm projekty spustíme příkaz elm package install elm-lang/mouse.

2.3.2 Elm-make

Nástroj elm-make použijeme v té chvíli, kdy chceme výslednou ELM aplikaci kompilovat do stránky HTML nebo do kódu JavaScript. V obou případech se ale vytvoří z kódu ELM kód JavaScript, akorát v případě kompilace do stránky HTML se tento kód vloží přímo do webové stránky HTML. Syntaxe pro použití je:**elm-make název_souboru.elm**. V tomto případě se nám ale automaticky vytvoří zkompilovaný soubor, jež se bude jmenovat index.html. Pakliže bychom chtěli vytvořit soubor s vlastním názvem a koncovkou .js namísto .html syntaxe bude následující:
elm-make název_souboru.elm –output název_souboru.js.

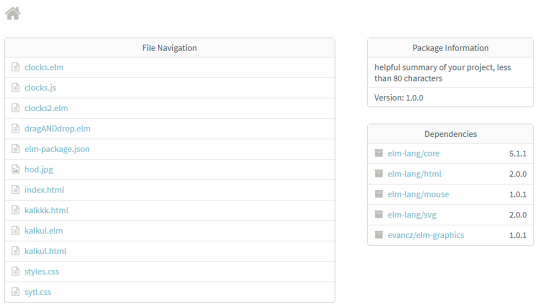


Obrázek 5: elm-make

V případě tohoto stylu kompilace se nám ale vypíší pouze závažné chyby, se kterými spuštěná kompilace ani neproběhne a skončí neúspěchem. V případě, že bychom chtěli být informování i o varováních na možné problémy, (které ale nejsou tak závažné aby neproběhla kompilace) musíme na konec příkazu elm-make napsat parametr –warn: **elm-make název_souboru.elm –output náze_zkompilovaného_souboru.js –warn**.

2.3.3 Elm-reactor

V případě, že nebudeme chtít ELM aplikaci hned kompilovat, ale budeme chtít mít možnost v reálném čase sledovat změny, které vytváříme a to, jak aplikace vypadá, jak se chová, jak reaguje, poslouží nám nástroj elm-reactor. Tento nástroj spouští vlastní localhost server, který nám umožňuje přejít na libovolný elm soubor a v reálném čase jej zkompilovat a zkontrolovat. Pro spuštění stačí ve vašem kořenovém adresáři s ELM soubory zadat pomocí cmd příkaz elm-reactor. V tu chvíli se spustí server, který bude naslouchat na adrese localhost a portu 8000.



Obrázek 6: elm-reactor

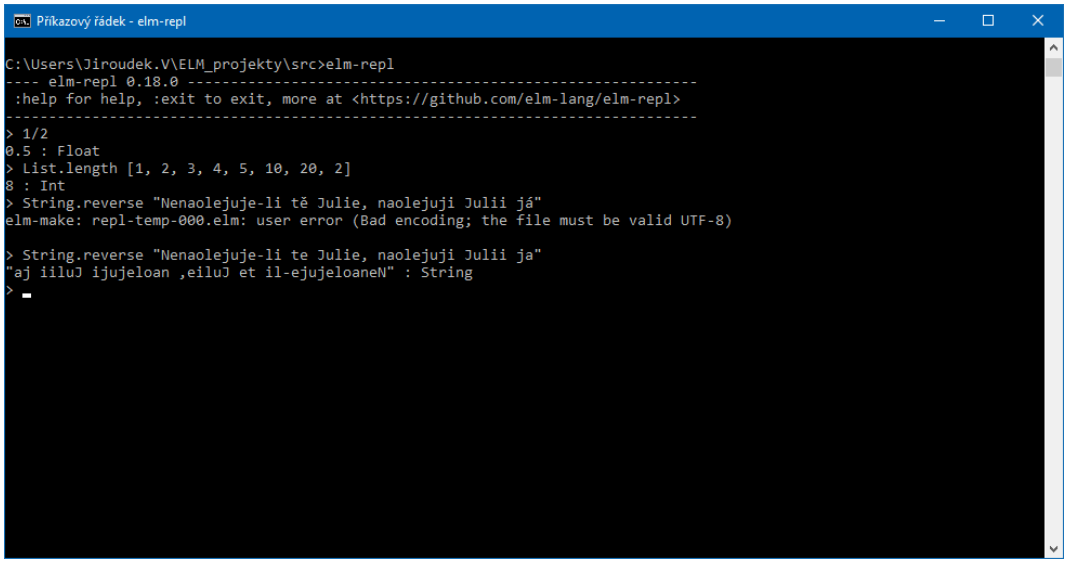
Příkaz elm-reactor je možné spustit s následujícími parametry:

- –port
- –address

–port nám umožňuje použít pro komunikaci jiný port, než defaultně nastavený port 8000. –address slouží k přejmenování adresy localhost adresou dle vlastního uvážení. Syntaxe by v těchto případech byla následující: **elm-reactor –port=libovolný_port, elm-reactor –address moje_adresa**.

2.3.4 Elm-repl

Poslední nástroj, který se jmenuje elm-repl se používá ke zkoušce jednotlivých ELM výrazů, kontrole jejich chování a výsledků. Například pokud budu chtít vyzkoušet matematický příklad 1 děleno 2 a budu chtít zjistit, zda výsledek bude zaokrouhlený na celé číslo, či nikoliv. Dále se pak dá otestovat chování u seznamu, jak bude reagovat textový řetězen na příkaz String.reverse atd..



Obrázek 7: elm-repl

Na příkladu můžeme vidět výhody nástroje elm-repl. Při provádění výrazu String.reverse máme okamžitě zpětnou vazbu. V případě použití v aplikaci bychom na toto přišli až při kompilaci.

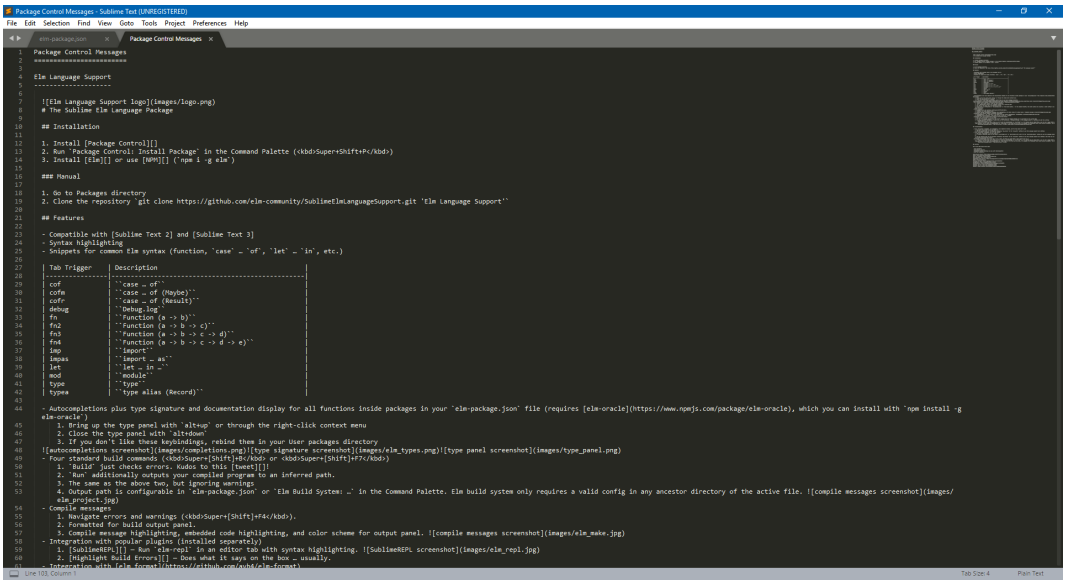
Pro funkčnost nástroje elm-repl je nutné mít nainstalovaný Node.js. Je to z toho důvodu, že nástroj elm-repl si ELM kód nejprve zkompiluje do JavaScriptu a až poté jej vyhodnotí. A Node.js nám poskytuje běhové prostředí pro tento zkompilovaný kód.

2.4 Vývojářské prostředí

Jelikož jazyk ELM nemá svoje vlastní vývojářské prostředí musíme pro vývoj aplikací jazykem ELM použít IDE třetí strany. Použít můžeme v podstatě jakýkoli textový editor. Nicméně pro větší komfort při psaní kódu je vhodné nakonfigurovat si své IDE (vývojové prostředí). Existují různé IDE vhodná k použití jazyku ELM. Např. Atom, IntelliJ, VS Code a další... Pro tvorbu jazykem ELM používám Sublime Text ve verzi 3.

Obrázek 8: Sublime text 3

Základní konfigurace tohoto textového editoru pro jazyk ELM je velmi jednoduchá. Nejprve pomocí nástroje Command palette (Tools -> Command palette...) nainstalujeme „Package Control“. Následně jej spustíme a pomocí něj vyhledáme položku „Elm Language Package“ a tu nainstalujeme.[5] Správně proběhlou instalaci poznáte tak, že se vám v Sublime Textu 3 otevře nová záložka, která bude vypadat přibližně takto:

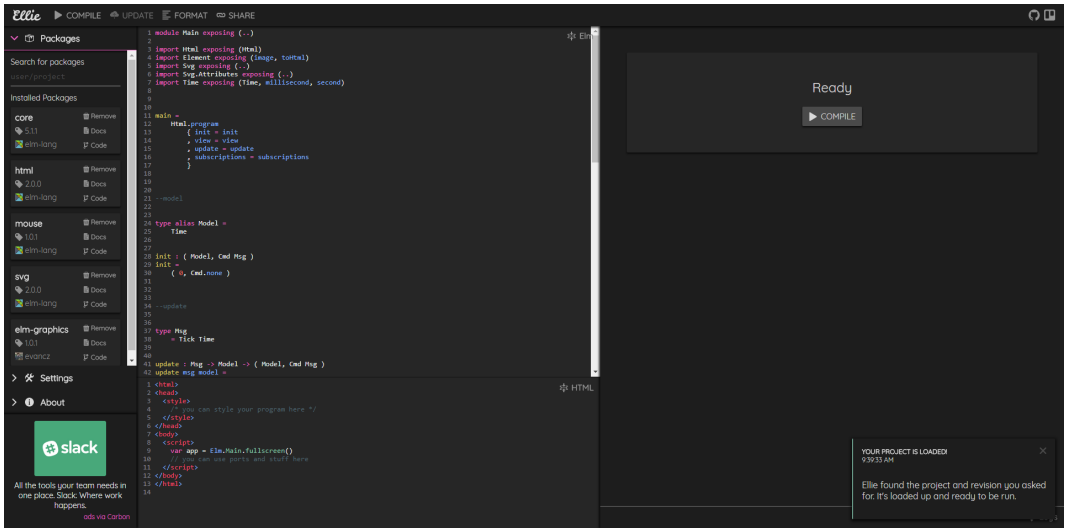


Obrázek 9: package control

Tím je naše IDE pro tvorbu jazykem ELM nakonfigurováno. Při vývoji aplikací se nám bude text barevně odlišovat, budou se nám zobrazovat návrhy atd...

V případě, že by nebyla možnost použití vlastního IDE, případně by byla poptávka pouze po vyzkoušení si jazyka ELM, je zde možnost použití online editoru. Online editory jsou v současné chvíli dostupné dva. První vytvořil sám Evan

Czaplicky při vytváření jazyka ELM a nalézt jej můžeme zde. Tento online editor je velice jednoduchý a určený pouze pro základní použití. Pokud bychom chtěli použít editor pro složitější příklady, můžeme zvolit online editor Ellie, který nám poskytuje možnost instalace nových balíčků, ukládání rozpracovaných projektů aj. Nalézt jej můžeme zde.



Obrázek 10: online editor Ellie

3 Syntaxe jazyka ELM

Každý programovací jazyk má svou vlastní syntaxi a sémantiku. ELM také poskytuje jednoduchou, ale silnou syntaxi pro psaní výrazů a vytváření aplikací. Níže budou popsány nejdůležitější části.

3.1 ELM architektura

Jazyk ELM má jednoduchou architekturu svých souborů. Struktura psaného kódu je skvělá pro opakované použití a udržovatelnost kódu v případě složitých webových aplikací. V programovacím prostředí jsou známé nástroje, například Redux (open-source JavaScriptová knihovna), které jsou inspirovány architekturou ELM. Základní vzorec architektury ELM je následující a dělí se na tři části:[6]

1. Model - obsahující stav aplikace
2. Update - popisující způsob aktualizace stavu aplikace
3. View - způsob zobrazení stavu aplikace jako HTML

3.2 Základy

3.2.1 Komentáře

Stejně jako v jiných programovacích jazycích, i v ELM lze použít komentáře.[7]

```
-- jednoradkovy komentar


{- viceradkovy komentar
   {- viceradkovy komentar muze byt vnoreny -}
-}
```

Příklad 1: komentáře v ELM

3.2.2 Aritmetika

Provádění aritmetický výrazů v ELM je jednoduché. ELM můžeme použít v podstatě jako kalkulačku.

Zadání	**Výsledek**
2 + 2	4 typ int
8 - 2	6 typ int
10 * 2	20 typ int
11 / 2	5.5 typ float
11 // 2	5 typ int

Tabulka 1: Aritmetika

3.2.3 Pravdivostní proměnné

Co se týče pravdivostních proměnných, tak oproti jiným jazykům je u ELM změna pouze v zápisu "nerovná se".

Zadání	**Výsledek**
not True	False
not False	True
1 == 1	True
1 /= 1	False
1 < 12	True

Tabulka 2: Pravdivostní proměnné[7]

3.2.4 Řetězce a znaky

Stejně jako v jiných programovacích jazycích máme i v jazyce ELM textové řetězce a znaky. Jejich syntaxe je klasická. Znak značíme **'a'** a řetězec **"textový řetězec"**. Práce s nimi je obdobná jako v jiných programovacích jazycích, například řetězce můžeme spojovat: **"jak "++ "se "++ "máš?"**.

3.3 Seznamy

Podobně jako ostatní programovací jazyky má i jazyk ELM seznamy. Každá položka seznamu musí nutně být stejného datového typu. Například seznam **['a','b','c']** je v pořádku a datový typ uložených hodnot je char. Naopak seznam **[1,'b',"ahoj"]** je špatně vytvořený a kompilace by nám skončila chybou, jelikož obsahuje tři různé datové typy (int, char, string).

Číselný seznam můžeme také vytvořit zadáním rozsahu např. **List.range 1 5** nám automaticky vytvoří seznam **[1,2,3,4,5]**. Pozor, toto funguje pouze u číselného seznamu. Další co můžeme se seznamy dělat, je kontrolovat jestli obsahují danou hodnotu. K tomu slouží **List.member**. Pro kontrolu délky seznamu existují dva příkazy. **List.isEmpty** a **List.length**. K čemu slouží už napovídá jejich název. Dalšími příkazy pro práci se seznamy pak jsou: **List.reverse**, **List.concat**, **List.partition**, **List.unzip**, **List.sort** a další..[7]

3.4 Řídící struktury

Co se týče řídících struktur, tak jazyk ELM používá klasickou podmínku "if". Ta musí v každém případě obsahovat větev "else"a obě větve nutně musí být stejného datového typu. Syntaxe v případě vnořených podmínek je následující:

```
if body > 5 then
   "Mas hodne bodu"
 else if body == 5 then
   "Ziskal jsi 5 bodu"
 else
   "Priste se vic snaz"
```

Příklad 2: podmínka if v ELM

Použít můžeme také příkaz "case", který slouží k nalezení shody porovnávaného a různých možností. Shody se vyhodnocují v zapsaném pořadí, takže je potřeba dávat pozor na zápis kódu.[8]

```
case barva of
        cervena ->
            "Vybral sis cervenou barvu"
        cerna ->
            "Vybral sis cernou barvu"
        zluta ->
            "Vybral sis zlutou barvu"
        modra ->
            "Vybral sis modrou barvu"
        _ ->
            "Vybral sis barvu, kterou neznam"
```

Příklad 3: case v ELM

Jako poslední možnost je znak _, který ošetřuje všechny zbývající možnosti. Psát jej musíme nakonec, protože pokud bychom ho napsali na začátek, case na něm vždy skončí.

3.5 Let výrazy

Klíčové slovo let definuje výrazy pro další použití ve výrazu za klíčovým slovem in. Tento zápis je užitečný, když se výraz výrazně zvětšuje a my jej chceme rozdělit na menší definice.[9]

```
let
  patnact =
    5 + 10


  sestnact =
    4 ^ 2
in
  patnact + sestnact
```

Příklad 4: case v ELM

3.6 Funkce

Funkce jsou pro aplikace ELM tak důležité, že ELM poskytuje pro jejich vytvoření neuvěřitelně jednoduchou syntaxi. Není potřeba použití žádného klíčového slova nebo složených závorek { }, jak je tomu běžné v jiných programovacích jazycích. Elm podporuje dva typy funkcí:[10]

1. Anonymní funkce
2. Pojmenované funkce

3.6.1 Anonymní funkce

Anonymní funkce, jak její název naznačuje, je funkce, kterou vytvoříme bez pojmenování:

```
\promenna -> promenna + 10
```

Příklad 5: anonymní funkce v ELM (1)

```
\x y -> x + y
```

Příklad 6: anonymní funkce v ELM(2)

Mezi zpětným lomítkem a šipkou se uvádí argumenty funkce a vpravo od šipky se určuje, co s těmito argumenty dělat.

3.6.2 Pojmenované funkce

Pojmenované funkce je nejpoužívanější prvek v ELM. Její syntaxe je následující:

```
odectiDva : Int -> Int
odectiDva x =
  x - 2
```

Příklad 7: pojmenovaná funkce v ELM(1)

Prvním řádkem této funkce je její popis. Tento popis je v ELM nepovinný, ale silně se doporučuje jej používat, jelikož nám přehledně sděluje záměr funkce. Zbytek příkladu je implementace funkce, která musí splňovat popis funkce. V tomto případě nám popis funkce říká, že se funkce jmenuje "odectiDva", jako vstupní argument jí bude celé číslo (Int) a výstupní hodnota bude také celé číslo. V případě, že budeme chtít funkci později použít pouze jí použijeme příkazem **odectiDva 5**. Další možnost, jak napsat pojmenovanou funkci je tato:

```
minus : Int -> Int -> Int
minus x y =
  x - y
```

Příklad 8: pojmenovaná funkce v ELM(2)

Tato funkce obsahuje dva vstupní celočíselné argumenty a jeden celočíselný výstup. Funkce, jež by neměla žádné argumenty je jednoduše konstanta. Důvod, proč se v ELM u funkce nepoužívají závorky, je ten, že funkce je vždy vyhodnocována zleva. Pokud bychom tedy chtěli pomocí dvou funkcí nejprve sčítat a

potom násobit musíme dát pozor na použití zápisu **vynasob 2 (secti 5 3)**. V tuto chvíli bychom volali funkci vynasob s parametrem dva a výsledkem druhé funkce secti.

Pro přehlednost zápisu funkcí se doporučuje využívat prvek, který se nazývá potrubí. V případě použití potrubí by zápis ilustrovaného součtu s násobením vypadal následovně:

```
3
    |> secti 5
    |> vynasob 2
```

Příklad 9: použití funkce s potrubím

V tomto případě je hodnota 3 předána částěčně aplikované funkci secti 5 a výsledek je poté předán částečně aplikované funkci vynasob 2. V tomto případě je použití potrubí nadbytečné, nicméně v komplexnějších případech velice zpřehlední kód.

4 Aplikace

Praktická část této bakalářské práce zahrnuje sadu vzorových webových aplikací, které jsou umístěné na webové stránce www.jirouv.wz.cz. Pro demonstraci bylo použito webového hostingu poskytovaného na webové stránce www.webzdarma.cz. Využito bylo hostingu bezplatného, tudíž můžeme u aplikací vidět přidanou reklamu.

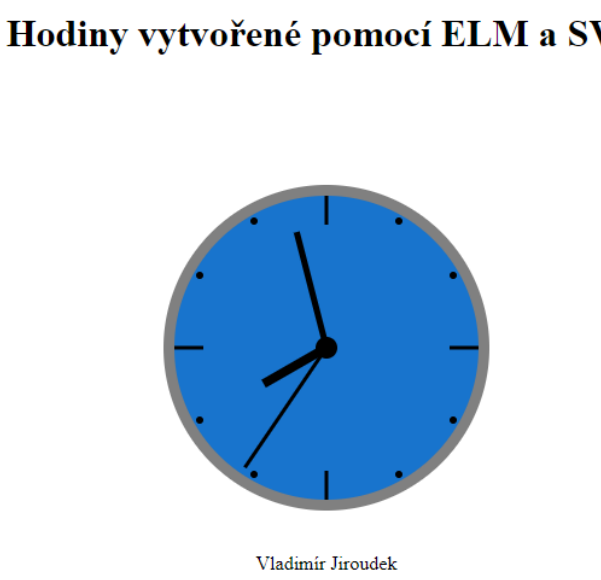
Webová stránka kromě samotných aplikací obsahuje i úvodní rozcestník a základní popis jazyka ELM. Aplikace byly vybrány tak, aby pomocí nich bylo možné demonstrovat některé z možností jazyka ELM. Je použito různých typů grafických formátování a importovány různorodé knihovny.

Všechny aplikace byly testovány na prohlížeči Internet Explorer, Edge, Google Chrome, Mozilla Firefox, Opera, mobilní Google Chrome a mobilní AdBlock Browser. Na žádném z nich nebyl zaznamenán jakýkoli problém s během aplikace nebo s jejím zobrazením.

Jakožto příloha je poskytnuto CD, na kterém jsou vypáleny všechny aplikace se zdrojovými kódy a přílohami.

4.1 Hodiny

Webová aplikace hodiny slouží ke zobrazení aktuálního času. Využívá k tomu klasických rafičkových hodin. Aplikace je určena pro umístění na webové stránky. Využívá knihovny Time a jejích možností práce s časem.



Obrázek 11: aplikace hodiny

4.1.1 Popis funkcí aplikace

Všechny funkce aplikace jsou obsaženy v souboru pojmenovaném hodiny.elm, nicméně je samozřejmě možné pro lepší přehlednost kód rozdělit do více souborů. V úvodu jsou naimportovány knihovny.

```
1 import Html exposing (..)
2 import Html.Attributes exposing (..)
3 import Svg exposing (..)
4 import Svg.Attributes exposing (..)
5 import Time exposing (..)
```

Příklad 10: import knihoven v aplikaci hodiny

Důležitým prvkem jsou zejména knihovny Svg a Time. Klíčové slovo exposing (..) znamená, že nevybíráme jen některé funkce knihovny, ale umožňujeme si práci se všemi. V další části kódu je zajištěna aktualizace v reálném čase s každou uplynutou sekundou. Poslední velice důležitou částí kódu jsou funkce pro stanovení umístění a směru každé z časových ručiček.

```
1  sekundyRuc : Time -> Svg Msg
2  sekundyRuc model =
3      let
4          uhel =
5              turns (Time.inMinutes model) - pi / 2
6
7          x =
8              toString (50 + 40 * cos uhel)
9
10         y =
11             toString (50 + 40 * sin uhel)
12     in
13     line [ strokeWidth "1px", x1 "50", y1 "50", x2 x, y2 y
           , stroke "black" ] []
```

Příklad 11: funkce sekundové ručičky v aplikaci hodiny

4.1.2 Grafika

Zajímavé na této aplikaci je, že její vzhled je řešen kompletně pomocí SVG (formát pro vektorovou grafiku) s mírnou pomocí CSS (pozicování, text, atd..). Svg knihovna musí být naimportována. Hodiny jsou dělány přehledně a účelně.

```
1  view : Model -> Html Msg
2  view model =
3      div []
```

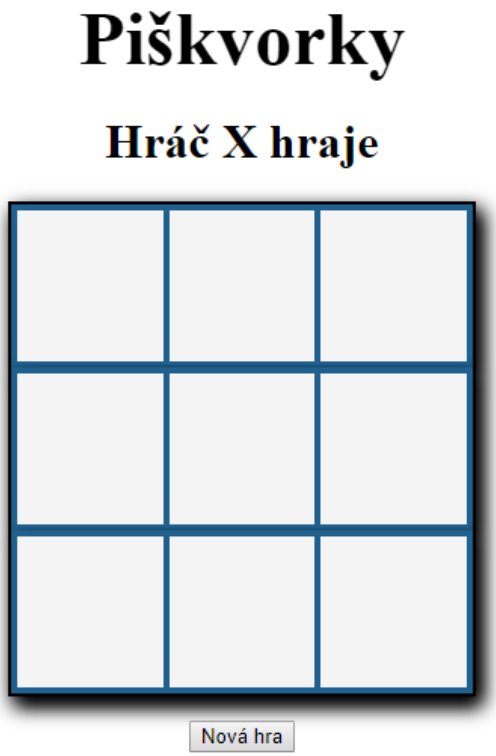
```
4           ...
5           , div [ stylHodin ]
6               [ svg
7                   [ viewBox "0 0 100 100", Svg.Attributes.
                        width "300px" ]
8                   [ circle [ cx "50", cy "50", r "45", fill
                        "#808080" ] []
9                   , circle [ cx "50", cy "50", r "42", fill
                        "#1874CD" ] []
10                  , circle [ cx "50", cy "50", r "3", fill
                        "#black" ] []
11                  , polyline [ fill "none", stroke "black",
                        points "50,8 50,16" ] []
12                  , polyline [ fill "none", stroke "black",
                        points "50,92 50,84" ] []
13                  , polyline [ fill "none", stroke "black",
                        points "8,50 16,50" ] []
14                  , polyline [ fill "none", stroke "black",
                        points "92,50 84,50" ] []
15                  , circle [ cx "15", cy "70", r "1", fill
                        "#black" ] []
16                  , circle [ cx "30", cy "85", r "1", fill
                        "#black" ] []
17                  , circle [ cx "15", cy "30", r "1", fill
                        "#black" ] []
18                  , circle [ cx "30", cy "15", r "1", fill
                        "#black" ] []
19                  , circle [ cx "85", cy "30", r "1", fill
                        "#black" ] []
```

```
20                  , circle [ cx "70", cy "15", r "1", fill
                      "#black" ] []
21                  , circle [ cx "85", cy "70", r "1", fill
                      "#black" ] []
22                  , circle [ cx "70", cy "85", r "1", fill
                      "#black" ] []
23
24                  , sekundyRuc model
25                  , minutyRuc model
26                  , hodinyRuc model
27                  ]
28          ...
29              ]
30          ]
```

Příklad 12: použití SVG v ELM

4.2 Piškvorky

Aplikace piškvorky je určena pro dva hráče, obsahuje herní pole 3x3, stav o tom, kdo je na tahu, případně kdo vyhrál a tlačítko sloužící ke spuštění nové hry. Aplikace se ovládá pouze myší.



Obrázek 12: aplikace piškvorky

4.2.1 Popis funkcí aplikace

V úvodu jsou naimportovány knihovny:

```
1 import Html exposing (..)
2 import Html.Attributes exposing (..)
3 import Html.Events exposing (..)
4 import String
```

Příklad 13: import knihoven v aplikaci piškvorky

Funkce update dále určuje v jakých případech se bude hra aktualizovat a co se bude dělat. Nejprve pomocí case zjistí, jestli hráč zvolil tlačítko nová hra, nebo umístil znak. V případě, že umístil znak kontroluje, jeslti již je vítěz. Pokud ano, nic se neprovede. V opačném případě se umístí znak, zkontroluje se případný vítěz, remíza nebo se přepne na druhý znak.

```
update : Msg -> Model -> ( Model, Cmd msg )
update msg model =
    case msg of
        NovaHra ->
            initModel

        UmistitZnak pozicex pozicey ->
            if model.jeVitez then
                model ! []
            else
                let
                    noveHerniPole =
                        bunka model pozicex pozicey

                    novyVitez =
                        kontrolaVitez noveHerniPole

                    jeRemiza =
                        if novyVitez then
                            False
                        else
                            jeNaTahu noveHerniPole

                    novyTah =
                        if novyVitez then
```

```
                        model.tah
                    else if model.tah == 'X' then
                        'O'
                    else
                        'X'
            in
            { model | herniPole = noveHerniPole, tah =
                novyTah, jeVitez = novyVitez, jeRemiza
                = jeRemiza } ! []
```

Příklad 14: funkce update v aplikaci piškvorky

Pro demonstraci níže vidíte funkci "bunka", která umisťuje nový znak.

```
bunka : Model -> Int -> Int -> List (List Char)
bunka model pozicex pozicey =
    List.indexedMap
        (\y radekk ->
            List.indexedMap
                (\x bunkaa ->
                    if x == pozicex && y == pozicey &&
                       bunkaa == ' ' then
                        model.tah
                    else
                        bunkaa
                )
                radekk
        )
        model.herniPole
```

Příklad 15: funkce bunka v aplikaci piškvorky

Všechny další funkce jsou přiloženy ve zdrojovém kódu na CD.

4.2.2 Grafika

Grafická stránka aplikace je vytvořena pomocí stylů CSS, které jsou vloženy přímo do kódu ELM. Tento způsob použití CSS je sice nejjednodušší, nicméně nám kód ELM zbytečně prodlužuje a zároveň zvětšuje soubor po kompilaci. V tomto případě o cca 4 kB.

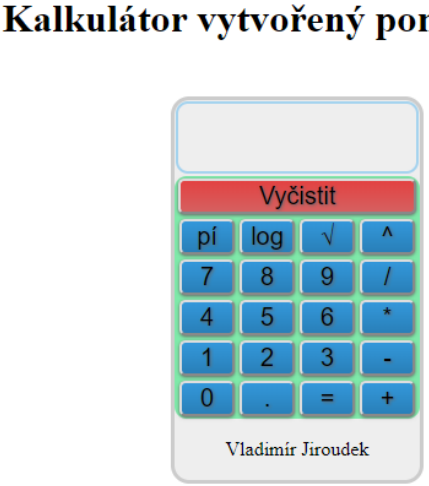
```
1
2  titulekstyl : Attribute Msg
3  titulekstyl =
4      style
5          [ ( "font-size", "50px" )
6          , ( "margin-bottom", "0" )
7          , ( "text-align", "center" )
8          ]
9
10
11 herniPolestyl : Attribute Msg
12 herniPolestyl =
13     style
14         [ ( "width", "310px" )
15         , ( "text-align", "center" )
16         ]
17
18
19 policekostyl : Attribute Msg
20 policekostyl =
21     style
22         [ ( "width", "100px" )
23         , ( "height", "100px" )
24         , ( "border-style", "solid" )
```

```
25              , ( "border-width", "2px" )
26              , ( "border-color", "#006797" )
27              , ( "background-color", "#F5F5F5" )
28              , ( "font-size", "72px" )
29              ]
30                  ]
31              ]
32
33  ...
```

Příklad 16: ukázka CSS v aplikaci piškvorky

4.3 Kalkulátor

Aplikace, jež funguje jako klasický kalkulátor, přičemž podporuje vstup od myši a klávesnice. Zahrnuje jak základní matematické operace, tak i odmocninu, číslo pí a logaritmus. Podporuje práci s mezivýsledky.



Obrázek 13: aplikace kalkulátor

4.3.1 Popis funkcí aplikace

Na začátku dojde ke klasickému importu knihoven a stanovení funkce main. Zajímavá je část Update. Nejprve si stanovujeme, jaký typ zpráv je možno předávat a poté určujeme práci s nimi.

```
1  type Msg
2      = Prazdno
3      | Deleno
4      | Krat
5      | Minus
6      | Plus
7      | Mocnina
8      | Odmocnina
9      | Logaritmus
10     | Rovno
11     | Desetina
12     | Nula
13     | Cislo Float
14     | Vycistit
15
16
17 update : Msg -> Model -> Model
18 update msg model =
19     case msg of
20         Prazdno ->
21             model
22
23         Vycistit ->
24             init
25
```

```
26          Cislo number ->
27              aktualizovatDisplay model number
28
29          Desetina ->
30              desetina model
31
32          Nula ->
33              nula model
34
35          Deleno ->
36              jedenVypocet model calculator.deleno
37
38          Krat ->
39              jedenVypocet model calculator.krat
40
41          Minus ->
42              jedenVypocet model calculator.minus
43
44          Plus ->
45              jedenVypocet model calculator.plus
46
47          Mocnina ->
48              jedenVypocet model calculator.mocnina
49
50          Odmocnina ->
51              jedenVypocet model calculator.odmocnina
52
53          Logaritmus ->
54              jedenVypocet model calculator.logaritmus
```

```
55
56          Rovno ->
57              rovno model
```

Příklad 17: update v aplikaci kalkulátor

Po zjištění toho, co po nás uživatel chce, je provedena daná funkce. Pro příklad uvádím funkce pro přidání nuly, desetinné čárky a funkci pro aktualizování displeje po výpočetní operaci.

```
1  nula : Model -> Model
2  nula model =
3      if String.isEmpty model.display || not model.
           moznoPridat then
4          { model
5              | display = "0"
6              , moznoPridat = False
7          }
8      else
9          { model | display = model.display ++ "0" }
10
11 ...
12
13 desetina : Model -> Model
14 desetina model =
15     if not (String.isEmpty model.display) && model.
           moznoPridat then
16         { model | display = pridejDesetinu model.display }
17     else
18         { model | display = "0.", moznoPridat = True }
19
20 ...
```

```
21
22 aktualizovatDisplay : Model -> Float -> Model
23 aktualizovatDisplay model number =
24     if model.moznoPridat then
25         { model | display = model.display ++ toString
               number }
26     else
27         { model | display = toString number, moznoPridat =
              True }
```

Příklad 18: funkce v aplikaci kalkulátor

4.3.2 Grafika

Grafická stránka aplikace je tvořena kombinací inline CSS a hlavně externího CSS souboru (stejně, jako se používá při tvorbě webových stránek pomocí HTML + CSS). Tento způsob nám dělá kód přehlednější, nicméně je problémový v tom, že jakmile jednou proběhne kompilace ELM, aplikace si soubor CSS "natáhne"a pokud bychom v něm pak něco pozměnili nedojde ke změně v reálném čase (v případě localhost serveru bychom museli smazat adresář elm-stuff, aby se kód CSS musel znovu "natáhnout"). Proto bych jej nedoporučoval.

```
1  .text{
2      text-align: center;
3
4  ...
5
6  .display {
7      border: solid 2px #A4D3EE;
8      border-radius: 10px;
9      font-size: 30px;
10     margin: 1px;
```

```
11      overflow: hidden;
12      height:55px;
13  }
14
15  .display-text {
16      color: #000;
17      margin: 0px 5px;
18      overflow: hidden;
19      text-align: right;
20  }
21
22  ...
23  }
```

Příklad 19: externí CSS v aplikaci kalkulátor

Pro správné použití v ELM aplikaci musíme externí CSS styl "importovat".

```
1  importStylu : Html Msg
2  importStylu =
3      let
4          tag =
5              "link"
6
7          attrs =
8              [ attribute "Rel" "stylesheet"
9              , attribute "property" "stylesheet"
10             , attribute "href" "styles.css"
11             ]
12
13         children =
14             []
```

```
15      in
16      node tag attrs children
17  }
```

Příklad 20: externí CSS v aplikaci kalkulátor

4.4 Formulář

Aplikace, jež demonstruje registraci nového uživatele, nicméně po mírných úpravách je možné jí použít jako libovolný formulář. Aplikace podporuje vstup z klávesnice a obsahuje kontrolu správně zadaného emailu (ve správném formátu). Aplikace si zadané údaje interně ukládá a je připravena na odeslání dat na server do databáze (backend development).

Obrázek 14: aplikace formulář

4.4.1 Popis funkcí aplikace

Na začátku jsou importovány knihovny a je spuštěn hlavní program.

```
1  import Html exposing (..)
2  import Html.Attributes exposing (..)
3  import Html.Events exposing (onClick, onInput)
4
5
6  main : Program Never Uzivatel Msg
7  main =
8      beginnerProgram
9          { model = init
10         , view = view
11         , update = update
12         }
```

Příklad 21: knihovny a main v aplikaci formulář

Jako model je vytvořen alias (slouží ke zjednodušení zápisu) Uzivatel, který obsahuje záznam o ukládaných datech.

```
1  --Model
2
3  type alias Uzivatel =
4      { jmeno : String
5      , email : String
6      , heslo : String
7      , registrovano : Bool
8      }
9
10
11 init : Uzivatel
```

```
12 init =
13     { jmeno = ""
14     , email = ""
15     , heslo = ""
16     , registrovano = False
17     }
```

Příklad 22: model v aplikaci formulář

Update poté obsahuje způsob ukládání daného uživatele.

```
1  type Msg
2      = UlozJmeno String
3      | UlozEmail String
4      | UlozHeslo String
5      | Registrace
6  update : Msg -> Uzivatel -> Uzivatel
7  update message uzivatel =
8      case message of
9          UlozJmeno jmeno ->
10             { uzivatel | jmeno = jmeno }
11
12         UlozEmail email ->
13             { uzivatel | email = email }
14
15         UlozHeslo heslo ->
16             { uzivatel | heslo = heslo }
17
18         Registrace ->
19             { uzivatel | registrovano = True }
```

Příklad 23: update v aplikaci formulář

4.4.2 Grafika

Grafická stránka aplikace je vytvořena pomocí inline CSS stejným principem jako v aplikaci piškvorky 4.2.2.

```
1  hlavicka : Attribute Msg
2  hlavicka =
3      style
4          [ ( "text-align", "center" )
5          , ( "position", "absolute" )
6          , ( "top", "50%" )
7          , ( "left", "50%" )
8          , ( "margin", "-250px 0 0 -50px" )
9          ]
10
11
12 formularStyl : Attribute Msg
13 formularStyl =
14     style
15         [ ( "border-radius", "10px" )
16         , ( "box-shadow", "10px 15px 10px #E8E8E8" )
17         , ( "background-color", "#EE8262" )
18         , ( "padding", "20px" )
19         , ( "position", "absolute" )
20         , ( "left", "50%" )
21         , ( "top", "50%" )
22         , ( "width", "300px" )
23         , ( "margin", "-150px 0 0 -150px" )
24         ]
25 ...
```

Příklad 24: CSS v aplikaci formulář

5 Srovnání ELM s JavaScriptem

Co se týče práce s ELM v porovnání s JavaScriptem panují mezi nimi určité podobnosti, ale i určité rozdíly. V následujících tabulkách je uvedeno pár příkladů.[12]

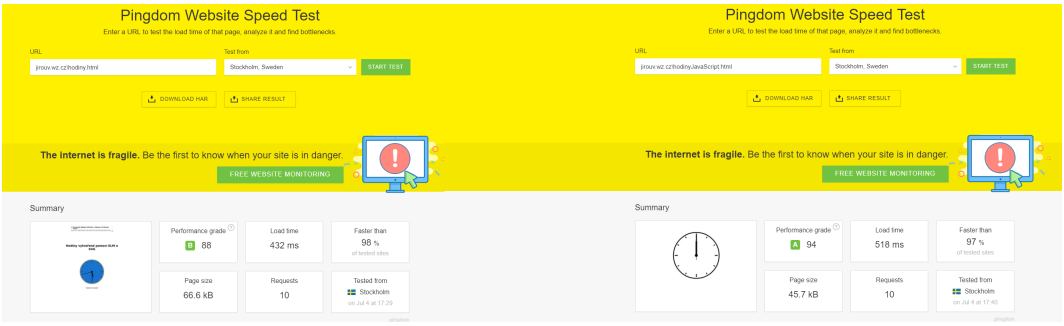
ELM	**JavaScript**
\x y -> x + y	function(x, y) return x + y;
max 12 13	Math.max(12, 13)
List.map sqrt numbers	numbers.map(Math.sqrt)

Tabulka 3: Funkce - ELM vs JavaScript

ELM	**JavaScript**
"ahoj"++ "123"	'ahoj' + '123'
String.length "ahoj"	'ahoj'.length
String.toUpper "ahoj"	'ahoj'.toUpperCase()
"ahoj"++ toString 123	'ahoj' + 123

Tabulka 4: String - ELM vs JavaScript

Za účelem porovnání byly vytvořeny analogové hodiny za pomoci JavaScriptu, SVG, Html a CSS[13]. Pomocí stránky tools.pingdom.com, která slouží k měření rychlosti webových stránek byly změřeny obě aplikace hodiny (jak v ELM, tak v JavaScriptu). Měření u každé webové aplikace proběhlo celkem 5x a poté byly výsledky zprůměrovány.



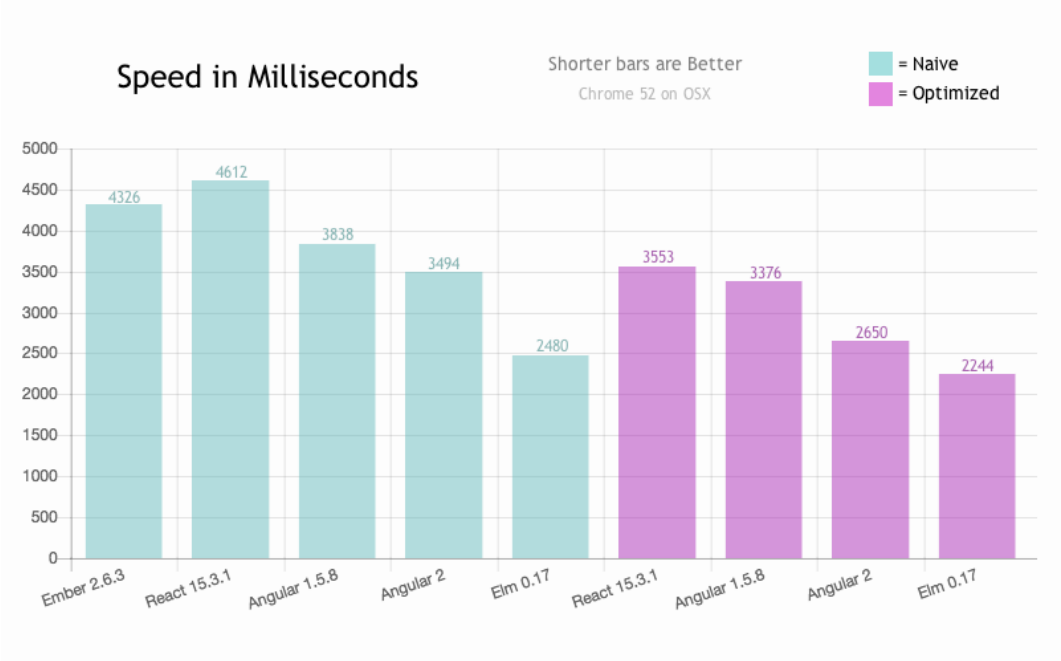
Obrázek 15: Rychlost ELM vs JS - aplikace hodiny

Jak si můžete povšimnout, i přes fakt, že webová stránka vytvořená v ELM je o cca 20kB (20% z celé stránky) větší, tak její čas načtení je nižší, než v případě JavaScriptu. Ve známce výkonu sice ELM aplikace zaostává (známka B oproti A), nicméně to je způsobeno tím, že při kompilaci ELM kódu se kompilují všechny balíčky apod., tudíž si nástroj pingdom nejspíš vyhodnotil, že její výkonnost by šla ještě vylepšit.

V průměrném hodnocení rychlosti stránky u ELM hodin vyšel výsledný čas potřebný pro načtení 430,6 ms, u JavaScriptových hodin 460,6 ms. Nějakých 30 ms rozdílu je sice zanedbatelných, ale pokud si uvědomíme, že se jedná o poměrně malou aplikaci, je rozdíl značný. V případě tvorby např. firemních velmi rozsáhlých aplikací (na něž je ELM převážně zaměřen) by byl rozdíl markantní.

5.1 Výhody ELM

Jak již bylo zmíněno výše, v porovnání rychlosti webových aplikací ELM jasně vede.



Obrázek 16: rychlost ELM[14]

Další nespornou výhodou jazyka ELM je jeho staticky typovaná kontrola. Dále pak poskytuje přehlednou architekturu kódu, která přispívá ke skvělé udržitelnosti a rozvoji dané aplikace. Na rozvoji jazyka ELM se neustále pracuje, takže vznikají stále nové knihovny (i od nezávislých vývojářů), což nabízí velký příslib do budoucna (v současné době existuje přibližně 554 balíčků). Stále více vývojářů začíná používat jazyk ELM a tudíž vzniká stále více zdrojů a návodů (od zadání této bakalářské práce až do jejího odevzdání vzniklo nespočet nových internetových zdrojů, bohužel pouze zahraničních).

ELM má velice dobrý kompilátor, který při kompilaci srozumitelně informuje o chybách a dokonce navrhuje řešení, jak je opravit. Tento fakt napomáhá vyhnutí se problémům při růstu aplikace. ELM kód je překládán do efektivního kódu JavaScript, což znamená, že se vývojář nemusí starat o změny v JavaScriptu, případně v prohlížečích.[15]

5.2 Nevýhody ELM

Stejně jako každý programovací jazyk i ELM má své nevýhody. Za nevýhodu považuji, že je ELM mladý programovací jazyk a tudíž poměrně nerozšířený. To má za následek omezené návody, zdroje a fóra. Pokud by někdo uměl HTML, CSS a chtěl vytvořit jednoduchou aplikaci v JavaScriptu, s nalezením zdrojů nebude mít problém. U ELM by to poměrně problémové bylo.

Zkompilované soubory bývají z pravidla větší, než v případě tvorby rovnou v JavaScriptu.

Za hlavní nevýhodu jazyka ELM považuji, že s postupným rozvojem a vydáváním nových verzí dochází k syntaktickým změnám. Například verze 0.16 (příklad aplikace v 0.16) byla syntakticky naprosto odlišná od verze 0.17, takže každý vývojář, který by chtěl použít novější verzi ELM by musel kód ručně předělat. Na druhou stranu mezi verzemi 0.17 a 0.18 (aktuální) byly rozdíly minimální a na internetových zdrojích jsem dokonce našel naprogramovaný převodník, který kód dokáže přepsat automaticky.

6 Závěr

Bakalářská práce se zabývala novým programovacím jazykem ELM, který je určen k využití při tvorbě webových aplikací. V teoretické části byly představeny obecné informace o jazyce ELM, o jeho vzniku, syntaxi, vývojovém prostředí, použití v prohlížečích apod. V praktické části byla vytvořena sada vzorových aplikací, na kterých bylo demonstrováno různorodé použití jazyku ELM. Jednalo se o použití SVG (vektorové grafiky), formuláře, početních operací, či možností ohledně času a her.

Jazyk ELM je programovací jazyk poměrně nový, teprve se rozvíjející s dobrým příslibem do budoucnosti. Dá se využít ke kompletnímu vytvoření webové stránky, případně pouze k vylepšení webových stránek stávajících.

Jelikož při vývoji nového programovacího jazyka zabere vždy několik let ustálení syntaxe, masové rozšíření mezi vývojáře, podpora ze strany firem apod. ještě si asi na plné využití jazyka ELM při tvorbě webových aplikací počkáme. Nicméně do budoucna je velkým příslibem vzestupná tendence co se týče množství vývojářů, dostupných knihoven a nápadů. Největším přínosem oproti jazyku JavaScript je ale jeho udržitelnost ve velkých a komplexních aplikacích.

Seznam použité literatury a zdrojů

[1] An introduction to Elm [online]. Czaplicki [cit. 2018-07-07]. Dostupné z: https://legacy.gitbook.com/book/evancz/an-introduction-to-elm/details

[2] GREENE, Kevin. Single-Page Web Apps in Elm. Linkedin [online]. 1. července 2016 [cit. 2018-07-04]. Dostupné z: https://www.linkedin.com/pulse/single-page-web-apps-elm-part-one-getting-started-new-kevin-greene/

[3] BOLF, Petr. Elm – Hello world on the map. Zdrojak [online]. 29.8.2016 [cit. 2018-07-04]. Dostupné z: https://www.zdrojak.cz/clanky/elm-uvod/

[4] NDUNG'U GATHUKU, Kevin. Building and Testing Web Applications with Elm. Semaphoreci [online]. Jun 28, 2017 [cit. 2018-07-04]. Dostupné z: https://semaphoreci.com/community/tutorials/building-and-testing-web-applications-with-elm

[5] Elm Language Support - Packages - Package Control. Packagecontrol [online]. 2017 [cit. 2018-07-04]. Dostupné z: https://packagecontrol.io/packages/Elm%20Language%20Support

[6] PORTO, Sebastian. Elm tutorial. Gitbook [online]. 2016 [cit. 2018-07-04]. Dostupné z: https://legacy.gitbook.com/book/sporto/elm-tutorial/details

[7] GOLDSTEIN, Max. Learn Elm in Y minutes. Learnxyinminutes [online]. 2018 [cit. 2018-07-04]. Dostupné z: https://learnxinyminutes.com/docs/elm/

[8] POUDEL, Pawan. Beginning Elm. Elmprogramming [online]. 2018 [cit. 2018-07-07]. Dostupné z: http://elmprogramming.com/case-expression.html

[9] Road to Elm - 'let' and 'in'. LambdaCat [online]. [cit. 2018-07-09]. Dostupné z: https://www.lambdacat.com/road-to-elm-let-and-in/

[10] REIMANN, Dennis. Elm Functions - Syntax, Piping and Currying. Dennisreimann [online]. Jan 21, 2016 [cit. 2018-07-04]. Dostupné z: https://dennisreimann.de/articles/elm-functions.html

[11] From javascript. Elm-lang [online]. 2018 [cit. 2018-07-09]. Dostupné z: http://elm-lang.org/docs/from-javascript

[12] ROMANK. Codepen.io. Codepen [online]. [cit. 2018-07-04]. Dostupné z: https://codepen.io/RomanKl/pen/xdNEMR

[13] CZAPLICKY, Evan. Blazing-fast-html-round-two. Elm-lang [online]. 30 Aug 2016 [cit. 2018-07-04]. Dostupné z: http://elm-lang.org/blog/blazing-fast-html-round-two

[14] GOOMAR, Rishi. Why I think Elm is the Future of Front End Development. Medium [online]. Oct 16, 2016 [cit. 2018-07-04]. Dostupné z: https://medium.com/@rgoomar/why-i-think-elm-is-the-future-of-front-end-development-21e9b091fa05

[15] An Introduction to Elm [online]. 2018 [cit. 2018-07-04]. Dostupné z: https://guide.elm-lang.org/

[16] MAIDA, Kim. Creating Your First Elm App: From Authentication to Calling an API. Auth0 [online]. 2018, August 04, 2016 [cit. 2018-07-04]. Dostupné z: https://auth0.com/blog/creating-your-first-elm-app-part-1/

Seznam obrázků

Seznam tabulek

Seznam příkladů

A Příloha

CD obsahující zdrojové kódy, obrázky a printscreeny, dále plné znění bakalářské práce v pdf.

B Příloha

Web: `http://www.jirouv.wz.cz`