DELTA – Střední škola informatiky a ekonomie s.r.o.

Ke Kamenci 151, PARDUBICE

Maturitní projekt:

Sbírka řešených a neřešených úloh z programování v Javascriptu

Jméno a příjmení: David Hrobař

Třída: 4.A

Studijní obor: Informační technologie

Školní rok: 2020/2021

Zadání maturitního projektu z informatických předmětů

Jméno a příjmení:	*David Hrobař*
Školní rok:	*2020/2021*
Třída:	4.A
Obor:	*Informační technologie 18-20-M/01*
Téma práce:	*Sbírka řešených a neřešených úloh z programování v JavaScriptu*
Vedoucí práce:	*RNDr. Jan Koupil, Ph.D.*

Způsob zpracování, cíle práce, pokyny k obsahu a rozsahu práce:

Cílem práce je vytvořit ucelenou sbírku úloh z programování v JavaScriptu tak, aby žák-začátečník mohl cvičit své programátorské schopnosti. Cvičení by měla být logicky seřazena do kapitol, mít postupně se zvyšující náročnost a kombinovat cvičení zaměřující se na jediný jev s komplexními úlohami. Cvičení by měla být do vysoké míry autonomní – tedy sama se opravující, resp. ohlašující chyby.

1. Žák podnikne rešerši za účelem nalezení existujících cvičení a tutoriálů k výuce čistého jazyka JavaScript a také vhodných nástrojů k jejich zkoušení (online prostředí i offline s použitím unit-testů).
2. Žák sestaví kurz jazyka tak, aby postupně přidávané jazykové konstrukce pokračovaly od nejjednodušších ke složitějším a jejich kombinování. Ke každému tématu žák připraví několik ukázkových použití a také cvičení. Neřešená cvičení a případně i ukázky budou využívat nástrojů nalezených v 1. úkolu tak, aby cvičení žákům dávala zpětnou vazbu, zda se podařilo úkol vyřešit, či nikoliv a v maximální míře též informaci, kde řešení selhává.
3. V závislosti na čase a tématech probíraných v nižších ročnících otestuje žák, pokud to bude možné, alespoň části své sbírky a formou osobní zpětné vazby nebo dotazníku zjistí, zda je zvolený přístup vhodný, a případně jej upraví.

Dokumentace práce bude odrážet tyto úkoly. V teoretické části bude popsáno, jak se jednotlivé části jazyka obvykle vyučují, jaký postup doporučuje autor této práce a proč, dále žák popíše vlastnosti zvoleného prostředí k výuce/testování, poté bude uvedeno několik ukázek

zadání cvičení a v poslední kapitole budou shrnuty zkušenosti s použitím, pakliže se tutoriál podaří alespoň částečně ověřit.

Stručný časový harmonogram (s daty a konkretizovanými úkoly):

- *Září*: Rešerše existujících řešení a vhodných nástrojů
- *Říjen*: Ladění testovacího prostředí, první jednoduchá cvičení
- *Listopad*: Návrh struktury sbírky, členění kapitol, plnění prvních částí
- *Prosinec*: Plnění sbírky obsahem
- *Leden*: Testování s uživateli (bude-li situace ve škole připouštět)
- *Únor*: Úpravy UI a navržených cvičení na základě výsledků předchozích testů, opakované testy, práce na dokumentaci projektu
- *Březen*: Práce na dokumentaci projektu

Prohlašuji, že jsem maturitní projekt vypracoval samostatně výhradně s použitím uvedené literatury

V Pardubicích dne …….

_____

Poděkování

Chtěl bych především poděkovat RNDr. Janu Koupilovi za skvělé vedení a motivaci při průběhu práce na projektu. Dále bych také chtěl poděkovat svým spolužákům za pomoc při testování funkčnosti a zabezpečení aplikace a také za podporu během celé práce.

Anotace

V dokumentu je představena a popsána práce zabývající se tvorbou webu obsahujícího programovací prostředí pro jazyk JavaScript, cvičení určených k plnění ve vytvořeném prostředí a administrativní části pro vytváření a spravování úloh.

Klíčová slova

Webová aplikace, ReactJs, Firebase, JavaScript, Cvičení v JavaScriptu,

Annotation

In the document we introduce and describe project that focuses on making an web application containing web programming interface for JavaScript language, exercises designed for solving in said interface and administrative part for creating and managing exercises.

Key Words

Web application, ReactJs, Firebase, JavaScript, exercises in Javascript

Obsah

Úvod

Cílem projektu je poskytnout začínajícím programátorům prostor kde zlepšovat a rozvíjet svoje schopnosti v jazyce JavaScript.

Konkurenci převážně tvoří stránky jako repl.it, jsfiddle nebo codewars [1]. Tyto weby také poskytují možnost online programovacího prostředí, ovšem s výjimkou codewars jsou vytvořeny ve stylu “playground”, kde si uživatel může kód vytvořit, spustit ho a sdílet s ostatními, avšak nemají žádný systém vlastních cvičení, ve kterých uživatel získá různá zadání s kontrolou a daným cílem, jak má kód vypadat a co má dělat.

1. Technologie

Použité technologie (níže uvedeno) byly vybrány s ohledem na povahu projektu. Krom zmíněných technologií byla také použita řada knihoven, které zprostředkovávají práci s uživatelským kódem.

1.1. ReactJs

React vytvořený společností Facebook je JavaScript open source knihovna určená pro front end vývoj. Převážně je používána k vývoji single-page aplikací. Mezi unikátní vlastnosti patři to, že je kód z většiny tvořen z komponent, které fungují podobně jako JavaScript funkce, ale vrací pomocí funkce render HTML kód. [2]

```
ReactDOM.render(<Greeter greeting="Hello World!" />,
document.getElementById('myReactApp'));
```

Obrázek 1 ukázka React komponenty

Celý vzhled aplikace je založen na ReactJs a knihovně Material-UI, vytvořené přímo pro použití s ReactJs, která obsahuje různé předpřipravené komponenty.

1.2. Javascript

Textový programovací jazyk, z většiny používaný k vývoji webových aplikací. Javascript je jediný jazyk použitý v aplikaci.

1.3. Firebase

Firebase je platforma od společnosti Google určená k vývoji webových aplikací. V projektu firebase zprostředkovává většinu Backend elementů, ku příkladu databázi či autentizaci. Mimo jiné poskytuje také možnost sledování návštěvnosti a zvětšování reklamy pro aplikaci. [3]

1.3.1. Firebase authentication

Firebase authentication poskytuje vývojáři backend servis v pohodě SDKs a knihoven pro snadné implementování ověřování identity uživatele. Umožňuje také využití různých sociálních sítí (např. Facebook) k ověření uživatele. [4] Projekt umožňuje uživateli přihlásit se jak pomocí mailové adresy, tak i s použitím účtu Microsoft.

1.3.2. Firebase firestore

Firebase firestore, NoSQL databáze, poskytuje možnost ukládat a synchronizování dat pro client i server side development. Data se zde ukládají ve formě dokumentů organizovaných v kolekcích. Firestore má také podporu pro off-line vývoj kdy vývojář může pracovat s daty i bez nutného připojení k síti. [5]

Firestore je v projektu využit k ukládání dat.

1.4. Vercel

Hosting systém od tvůrců frameworku Next.js, který také podporuje hostování ReactJs aplikací. Aplikace se zde nasazují velmi jednoduše a intuitivně, jsou přímo napojeny na GitHub, kde při každé nové změně kódu proběhne i update stránky.

1.5. InterpreterJS

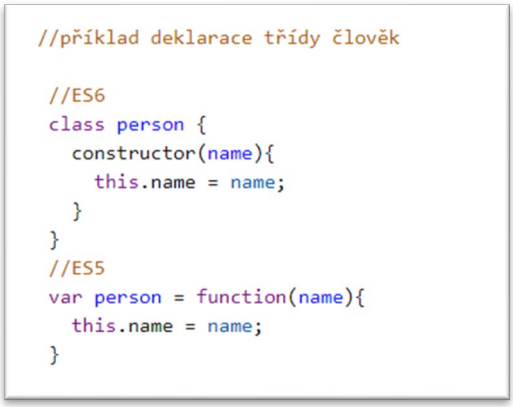
InterpreterJS, open source knihovna zajišťující bezpečné a ovládatelné spouštění jakéhokoliv kódu, který je izolovaný od originálního kódu aplikace. InterpreterJS je jednou z nejdůležitějších součástí projektu, poněvadž do jisté míry zprostředkovává programovací prostředí a jeho zabezpečení.

2. Problematika webového programování

Velkým problémem či limitací je krom zabezpečení, což je kapitola sama o sobě, také rychlost vykonávání samotného kódu. Výkonová limitace je především následek nedostatku paměti prohlížeče. Samozřejmě je jasné, že kód prováděný v prohlížeči bude o dost pomalejší nežli kód vykonaný procesorem. Kód může být stokrát ne-li víckrát pomalejší, avšak tento velký rozdíl nejde jen tak poznat. Nastává ve chvíli, kdy musíme provádět náročné nebo obsáhlé operace. Problém s výkonem by tedy mohl nastat ve chvíli, kdy náš kód obsahuje kolem 4 000 řádků. Nemusíme se proto obávat výrazných úpadků ve výkonu aplikace pro malá začátečnická cvičení.

Různí uživatelé používají různé prohlížeče. Náš cíl je, aby aplikace fungovala na co největším množství prohlížečů. Většina prohlížečů obsahuje svůj vlastní Javascript engine a každý prohlížeč podporuje jazyk do určité míry. Převážně se mluví o podpoře ECMASript standardu, kterým je JavaScript definován a tato standardizace zajišťuje, aby webové stránky fungovaly i v různých prohlížečích. Pro aplikaci je to velmi důležité, s příchodem nové verze ECMASript může plná implementace nových charakteristik do každého prohlížeče trvat i pár let. Může dojít tedy k tomu že se např. s novou verzí standardu objeví syntaxe které budou ihned používány v Javascriptových aplikacích, ale prohlížeče ji nějakou dobu nebudou podporovat. Uživatel by tak nemohl tuto novou syntaxi použít v našem projektu, i když je používaná a známá několik měsíců. V aplikaci se chceme vyhnout tomu, aby uživatel byl nucen používat zastaralou verzi a neměl možnost využít nové vlastnosti jazyka.

Projekt využívá knihovnu BabelJS, určenou právě k převádění nejnovějšího kódu na jeho starší verze, které jsou kompatibilní s většinou prohlížečů. Uživatelský kód je proto vždy před spuštěním převeden na starší verzi, bezpečně podporovanou. Uživatel má tak možnost používat nejnovější funkce jazyka bez ohledu na to jaký prohlížeč právě používá.



Obrázek 2 rozdíl mezi ES5 a ES6

3. Zabezpečení

Zabezpečení aplikace, z důvodu její povahy, je velice důležité téma. Vytvoření místa, kde ve vaší aplikaci může libovolný uživatel psát libovolný kód, je stejné jako implementace obřího tlačítka "rozbij mě a spoustu věcí kolem". Uživateli není možné poskytnout takovou možnost bez jakékoliv kontroly nebo omezení kódu který spouští. Tato omezení přicházejí ve formě skrytí přístupu k Web APIs např. Console Api, DOM apod. Některé z těchto Web APIs jsou také používané při samotném vytváření kódu. Nejspíše nejpoužívanější console.log(). Potřebné API calls jsou tedy nahrazeny objekty přímo v našem prostředí, které se chovají stejně jako originální, ale nemají přístup ven z prostředí.

Memory bomb také známá jako fork bomb je způsob útoku, který se neustále replikuje s cílem zatížit nebo znepřístupni služby webu. Útoku typu memory bomb se předchází pomocí pravidelného převádění kódu do JSON řetězce a při případném přesáhnutí velikosti daného řetězce dojde k přerušení provedení kódu.

Infinte loop je smyčka, která, jak její jméno naznačuje, bude probíhat napořád. Těmto smyčkám se chceme při programování vyhnout, protože mohou jednoduše shodit celý program. V online prostředí, ale nemusejí být vždy jenom chybou uživatele, mohou také být použity s úmyslem přetížit a shodit celou webovou stránku. Popsaný problém částečně řeší to, že všechny výpočty probíhají přímo u uživatele. Uživatel tedy maximálně přetíží svojí instanci stránky a serveru nijak neuškodí. Takové řešení ovšem není dostačující, nekonečným smyčkám chceme předejít kompletně i když nejsou vytvořeny se špatnými úmysly. Řešení je v podstatě jednoduché, při každém spuštění kódu kontrolujeme, jak dlouho probíhá určitá funkce či smyčka. Pokud narazíme na takovou část kódu, jenž se neustále opakuje po určenou dobu, přerušíme jeho vykonání

a uživateli vrátíme error „RangeError: Maximum call stack size exceeded“ naznačující že prohlížeč nemůže ustát takový nával argumentů. [6]

```
while(true)
{
  console.log(5)
}
```

Obrázek 3 příklad nekonečné smyčky

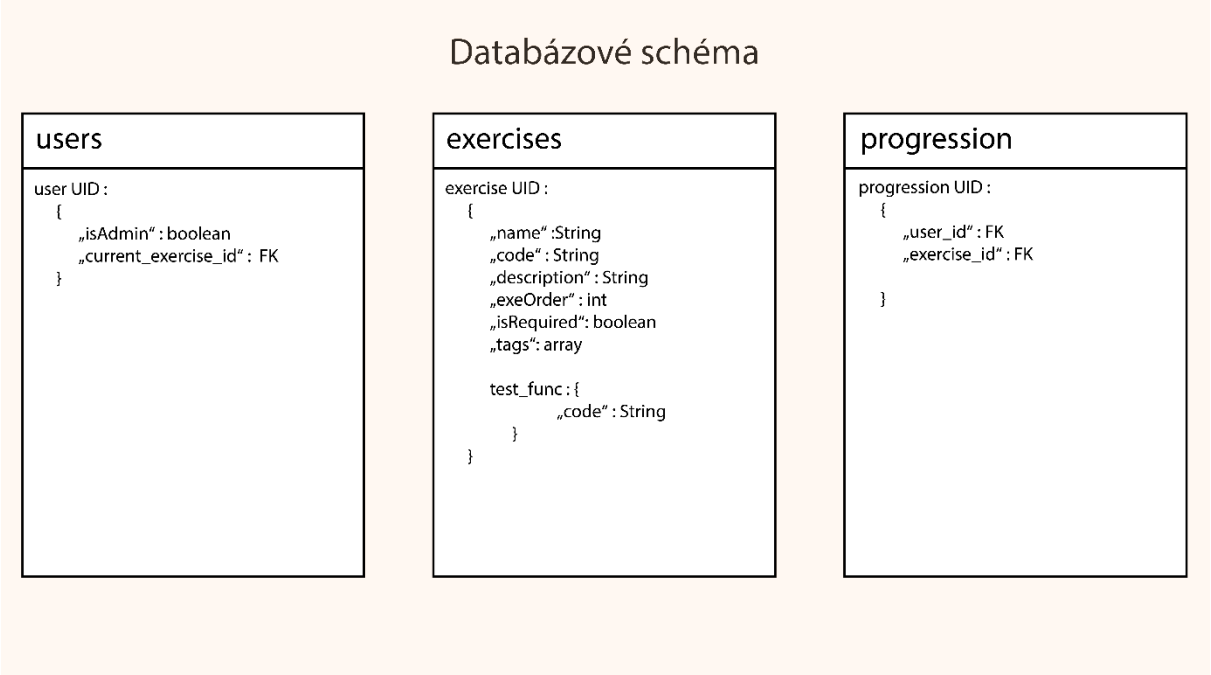
4. Databázový systém

Pro ukládání dat v projektu je využita NoSQL databáze firebase firestore. Data se zde ukládají do dokumentů, které následně řadíme do kolekcí. Databáze obsahuje 3 kolekce, users, exercises a progression. Při vytvoření nového dokumentu se vždy vygeneruje unikátní identifikátor, který je následně použit jako název daného dokumentu.

Kolekce uživatelů obsahuje pouze dvě pole, „isAdmin“ podle kterého jsou uživateli umožněny administrativní operace v aplikaci a currentExercise_id obsahující cizí klíč odkazující na dokument v kolekci s úlohami. Do kolekce uživatelů nemusíme ukládat data jako email, čas vytvoření účtu nebo poslední přihlášení, protože nimi nijak nepracujeme a služba firebase authentication zprostředkovávající ověřování uživatelů tato data ukládá zvlášť ve svém vlastním systému. Pokud by nastala chvíle, kdy musíme nějak pracovat s těmito daty, tak je lehce můžeme převést ze systému firebase authentication do naší databáze.

Kolekce úloh obsahuje základní informace jako jméno, zadání a pořadí úlohy. Kolekce také obsahuje pole s výsledným kódem úlohy, tento kód je vždy vytvořen autorem úlohy, nikoliv uživatelem. Výsledný kód je pouze používán při testování uživatelského řešení úlohy. V kolekci se také nachází podkolekce s testovacím kódem, který je také využívám při testování úloh.

Kolekce postupu obsahuje dva cizí klíče odkazující na uživatele a úlohu. Data z kolekce jsou využita na sledování splněných úloh u každého uživatele.

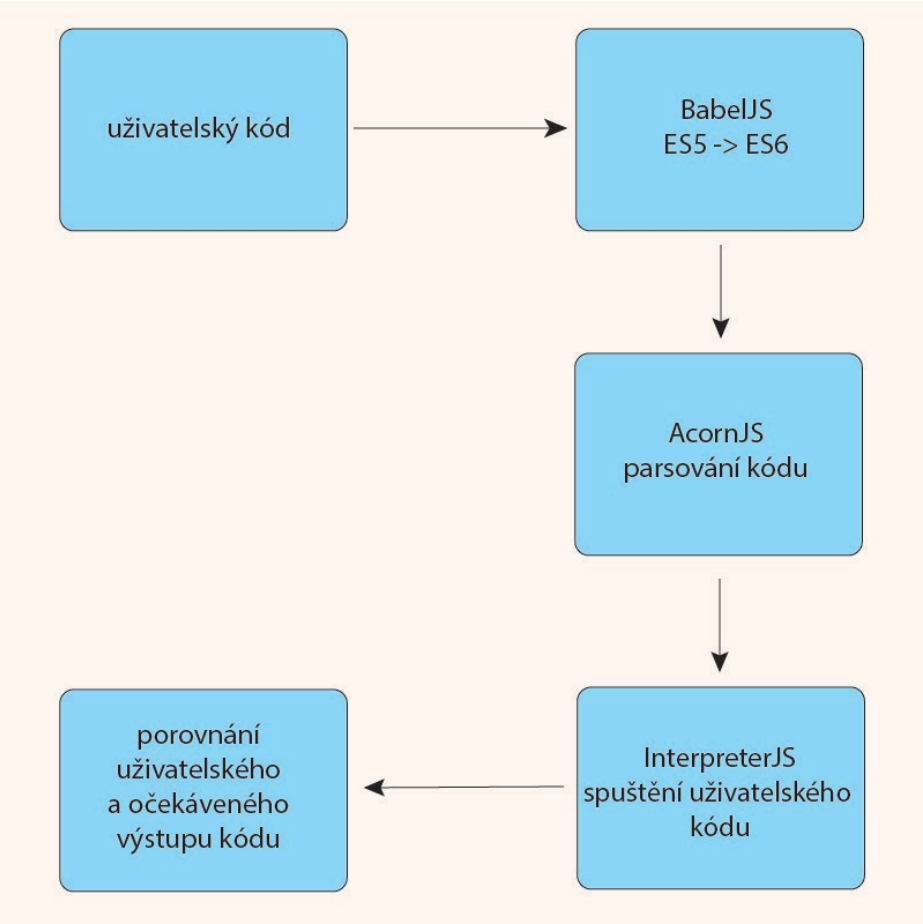


Obrázek 4 schéma databáze

5. Spouštění kódu

Uživatelský kód projde před spuštěním a testováním několika procesy přizpůsobení, tak aby byl bezpečný a mohl se plynule vykonat. Prvním procesem, kterým si kód projde je textová transformace z ES6 na ES5 pomocí BabelJS. Tento krok zajistí bezchybné spuštění kódu ve většině prohlížečů. Následujícím krokem je parsing daného kódu. Parsing je velmi důležitý, otevírá nám spoustu možností, jak s kódem zacházet. Například je mnohem jednoduší objevit a zabránit nekončícím smyčkám. Předposledním krokem je spuštění. Pokud uživatel pouze spustil kód, naše interakce s ním zde končí a výsledek, popřípadě error je zobrazen v konzoli. Pokud odevzdal své řešení úlohy, přichází poslední krok, ve kterém testujeme jeho řešení. Úlohy dělíme na dvě varianty. Začátečnické, tento typ je převážně jednoduchý na splnění, neobsahuje žádnou složitou logiku či využití algoritmů a jeho výstupem je vždy pomocí console.log(). Je pro nás velmi složité testovat nekomplexní úlohy s jednoduchým řešením. Jejich výsledky porovnáváme tedy s výsledky originálního řešení, které vzniká společně se zadáním. Druhá varianta obsahuje komplexnější úlohy, řešení většinou obsahuje práci s funkcemi, objekty či třídami. Pokročilejší úlohy jsou perfektní k testování, mohou přebírat různé parametry a vždy vrací nějakou hodnotu. Při vytváření komplexnějších úloh autor zavolá danou funkci s různými parametry, které by chtěl otestovat u uživatele. Testujeme, jestliže uživatelský kód, kterému předáme stejné parametry, vrátí také stejnou

hodnotu, jako kód vytvořený autorem. V případě že se výsledky neshodují, se uživateli vypíše, jaké parametry jeho kód nezpracoval správně. Uživatel má poté prostor svoje řešení upravit tak, aby tyto proměnné správně zpracoval. Pakliže je jeho řešení správné, dostane upozornění, že splnil úlohu a může pokračovat v dalším zadání.



Obrázek 5 životní cyklus uživatelského kódu

6. Funkce systému

6.1. Administrativní část

6.1.1. Vytváření úloh

Stránka určená pro vytváření úloh je vytvořena komponentou stepper, která postupně ukazuje formulářem pomocí očíslovaných kroků. Formulář je tedy rozdělen do tří částí. První část formuláře je zaměřená na popis dané úlohy, Název a zadání. Druhá část se věnuje pouze řešení. Řešení je potřeba převážně z důvodu testování. Autor má zde tedy vlastní prostředí, kde může

kód k úloze vytvořit a sám ho otestovat tak aby jeho výsledná forma odpovídala přesně zadání. Poslední kus formuláře je určen k vytvoření testů pro úlohu. Autor zde např. zavolá několikrát svoji funkci s různými parametry. Po vytvoření se úloha zobrazí ve výběru a autor může znovu vyzkoušet její funkčnost, ale už jako uživatel. [6]

6.1.2. Správa úloh

Rozhraní pro správu úloh je stejné jako rozcestník úloh pro uživatele. Na stránce v administrativní verzi se ale objevují nové funkce pro editování, odstraňování, měnění pořadí a vytváření úloh. Editace úloh má stejné prostředí jako stránka pro vytváření. Formulář obsahuje předvyplněná data zvolené úlohy, které správce může následně upravovat. Odstraňování je implementované přímo v rozcestníku jako jednoduché tlačítko. Řazení úloh se nachází také v rozcestníku místo v editační stránce. Správce může změnit pořadí úlohy výběrem a tažením. Po přemístění úloh dle uvážení správce se zobrazí tlačítko v pravém dolním rohu na uložení změn v pořadí úloh.

6.2. Uživatelská část

6.2.1. Autentizace

Ověřování uživatelů zprostředkovává firebase authentication. Při registraci uživatele proběhne vytvoření záznamu jak ve firebase authentication, tak v cloud firestore. Firebase authentication obsahuje data o poskytovateli, v našem případě, pokud je uživatel zaregistrován pomocí emailu či pomocí služeb Microsoft, čas vytvoření účtu, poslední přihlášení a unikátní uživatelský identifikátor. Identifikátor je také uložen v databázi v kolekci uživatelů.

Pro přístup do celé aplikace musí být uživatel přihlášen ke svému účtu. Kontrolu přihlášeného uživatele zajišťuje React Context API, které umožňuje globálně sdílet data mezi komponenty stránky bez toho abychom je museli výslovně předávat každému komponentu. [7]

6.2.2. Programovací prostředí

Programovací prostředí je nedílná součást celé aplikace. Rozhraní tvoří editor a výstupní konzole. Editor je navržen tak aby fungoval a vypadal jako vývojové prostředí. Vložený text v editoru je stylovaný po vzoru vývojového prostředí Visual Studio Code. Pro tyto styly je využita knihovna CodeMirror, která nabízí bohatý výběr pro stylování textu po vzoru již zmíněného prostředí. Knihovna také nabízí různé funkce jako je například autocomplete.

Pod výstupní konzolí uživatel najde zadání toho, jak by měl program fungovat. Najde zde i dvě tlačítka, jedno pro spuštění kódu a druhé pro odevzdání. Tlačítko pro spuštění převezme kód z editoru, vykoná ho a do konzole vypíše výslednou hodnotu, popřípadě chybové hlášení a řádek s výskytem chyby. Tlačítko určené pro odevzdání úlohy také spustí uživatelský kód, avšak před samotným spuštěním za kód přidá svůj vlastní text v podobě testovacích funkcí. Výsledek pak porovnává s očekávaným výsledkem, který je vytvořen autorem úlohy s aktuální úlohou. Na základě výsledku se poté objeví upozornění sdělující, že je kód správný a splňuje zadané požadavky nebo v opačném případě, že nesplňuje zadání a vypíše se na jakém testu selhal. V případě selhání dostane uživatel tedy jakousi nápovědu, která mu na rozdíl od prosté zprávy „neprošel“ dá směr, jak svůj kód upravit, aby splňoval zadané požadavky.

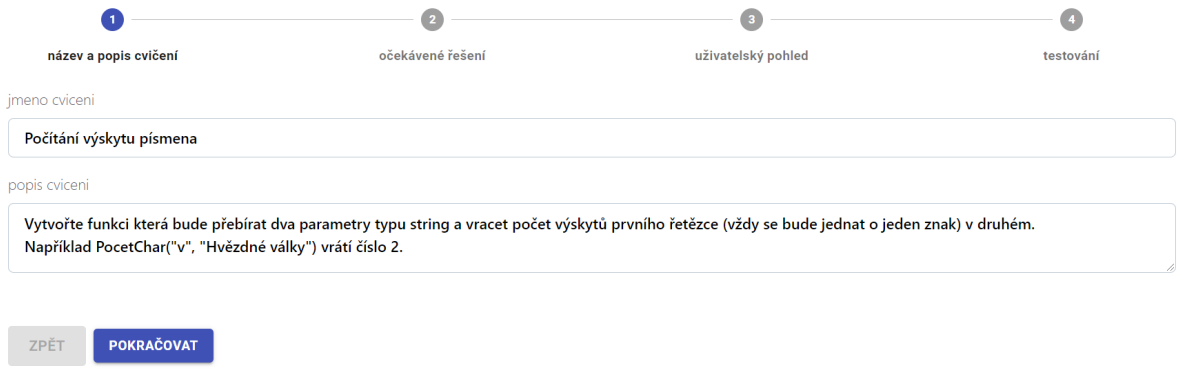
7. Grafická ukázka

Ukázka spočívá pouze pro předvedení již popsaných funkcí systému.

7.1. Vytvoření úlohy

7.1.1. Název a popis cvičení

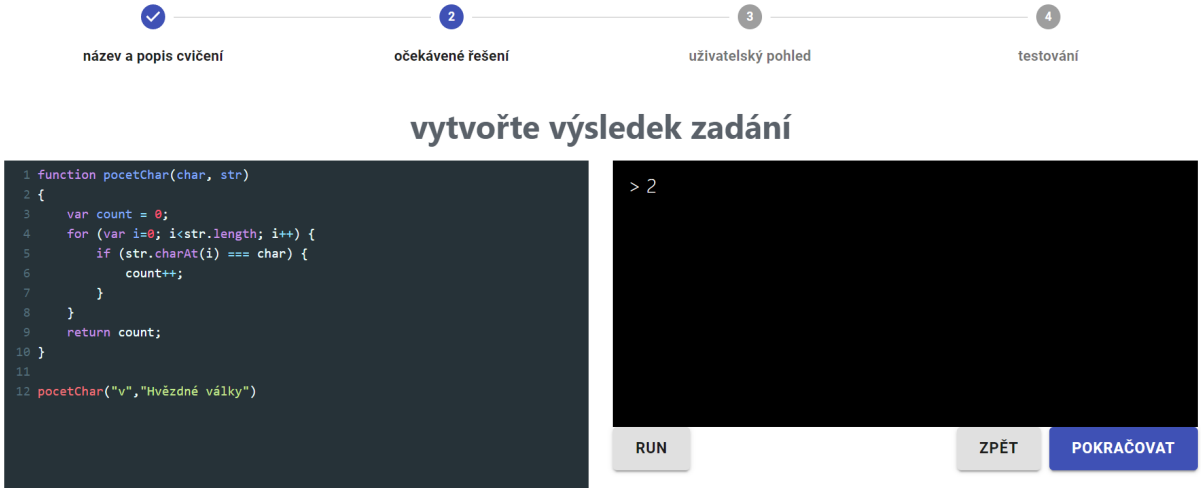
V prvním kroku se zadávají základní informace o cvičení. Název cvičení a samotný popis, tj. zadání pro vypracování výsledného kódu, jak by měl kód pracovat, jeho funkce a popřípadě příklad.



Obrázek 6 první krok při vytváření úlohy

7.1.2. Vytvoření výsledku

V druhém kroku se očekává řešení úlohy autorem. Autor zde může také dát důraz na různé prvky cvičení které by uživatel nemusel vždy očekávat a řešit je. V našem příkladě by to mohla být citlivost na malé a velké znaky.



Obrázek 7 druhý krok při vytváření úlohy

7.1.3. Vytvoření pohledu uživatele

V předposlední kroku má Autor možnost připravit uživatelský pohled. Kód, který uživatel uvidí ve svém prostředí při plnění úlohy.

Obrázek 8 třetí krok při vytváření úlohy

7.1.4. Testování řešení

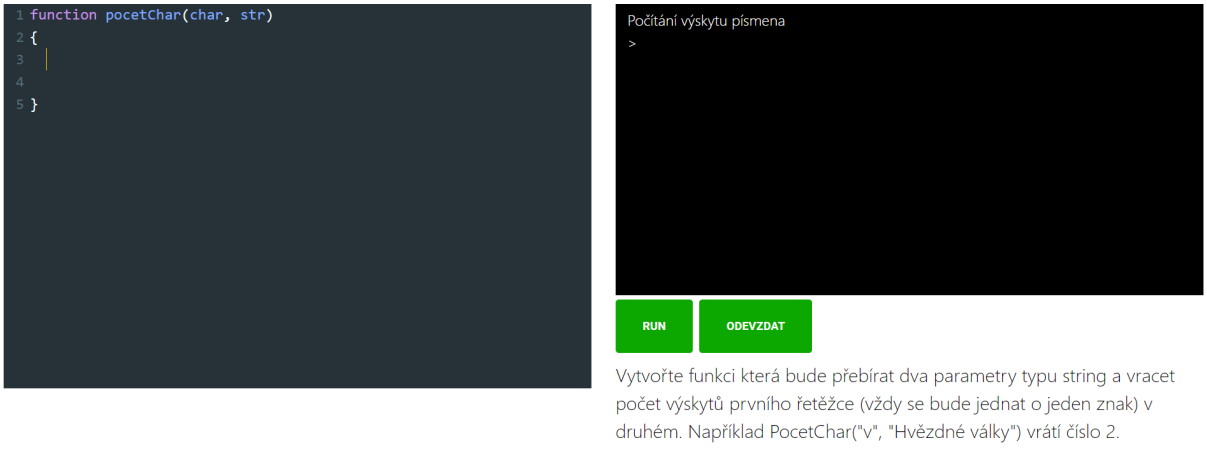
V posledním kroku autor vytvoří testování pro svoji úlohu. V našem případě obsahuje řešení funkci, autor ji zavolá s různými parametry. Stejné parametry budou testovány na uživatelském kódu. Pokud by řešení obsahovalo třídy, tak autor bude volat metody třídy či logovat její vlastnosti. Po vytvoření se úlohu zobrazí v rozcestníku.

Obrázek 9 čtvrtý krok při vytváření úlohu

7.2. Splnění úlohy

7.2.1. Prvotní pohled

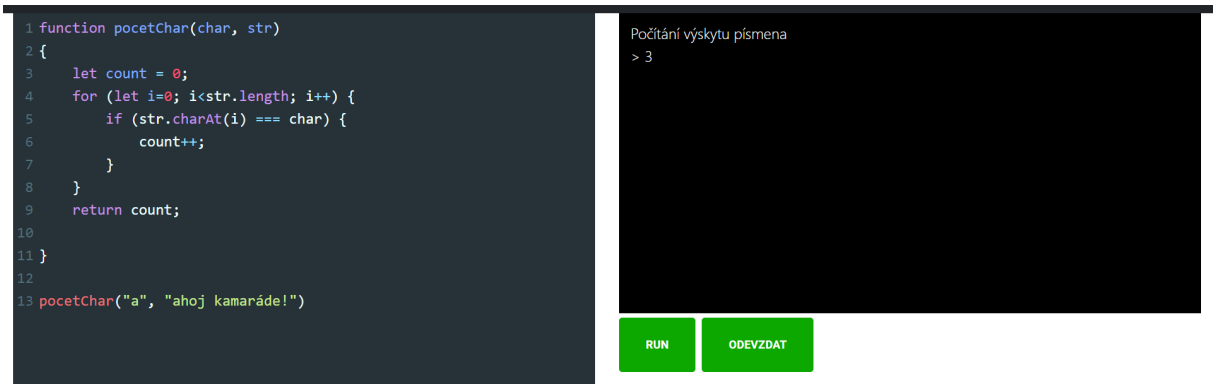
Při prvním spuštění úlohy se uživateli zobrazí prostředí pro psaní kódu, výstupní konzole a zadání pro danou úlohu.



Obrázek 10 první pohled uživatele na programovací prostředí

7.2.2. Spouštění kódu

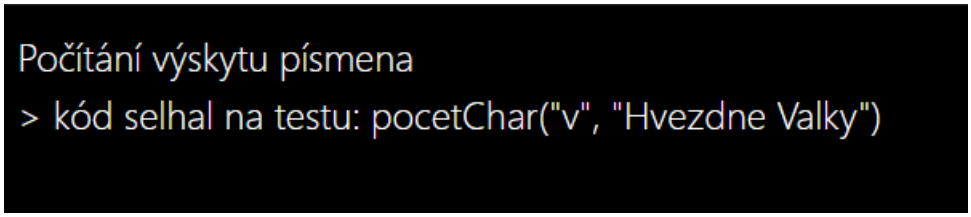
Uživatel má možnost v průběhu tvoření svého řešení spouštět svůj kód. V konzoli se vždy objeví výstupní hodnota nebo error pokud došlo k nějaké chybě či překlepu v řešení.



Obrázek 11 spuštění uživatelského kódu

7.2.3. Odevzdání

Pokud je uživatel jistý že jeho kód splňuje zadání má možnost ho odevzdat a získat zpětnou vazbu v podobě oznámení splnění či varování že jeho řešení selhává na nějakém z testů.



Obrázek 12 výpis chyby v konzoli

8. Firebase cloud functions

Cloud functions umožňují vývojáři hostovat a spouštět JavaScript kód v privátním Node.js prostředí. Cloud functions do jisté míry nahrazují backend server a pomocí Firebase SDK jsou tyto funkce použitelné ve webovém prostředí jako by byly vytvořeny na straně klienta. Na rozdíl od běžné REST API se k serveru nemusí přistupovat pomocí https requestů.

8.1. Aktualizace postupu

Funkce je vždy volána při dokončení cvičení. Vytváří dokument v kolekci progression s uživatelským identifikátorem a identifikátorem daného cvičení.

8.2. Aktualizace aktuálního cvičení

Funkce je volána při úspěšném splnění cvičení. Aktualizuje pole current_exercise v dokumentu s uživatelskými daty. Pole je aktualizováno s identifikátorem následujícího cvičení.

8.3. Hydratace uživatele

Funkce se zabývá vytvořením dokumentu v databázi s uživatelskými daty při prvotním přihlášení uživatele. Využívá event onCreate, který se vždy spustí při vytvoření záznamu účtu ve firebase authentication.

8.4. Vytvoření úlohy

Při zavolání vytvoří záznam v kolekci exercises s předanými daty. Pro vytvoření úlohy musí uživatel vlastnit roli admin.

8.5. Aktualizace úlohy

Funkce přebírá data o úloze která následně aktualizuje v příslušném dokumentu. Pro aktualizaci úlohy musí uživatel vlastnit roli admin.

8.6. Smazání úlohy

Funkce pomocí předaného ID smaže daný záznam úlohy v kolekci exercises. Uživatel musí vlastnit roli admin.

8.7. Získání dat o úloze

Funkce vrací kompletní data o úloze vlastnící předané id.

8.8. Získání dat úloh

Funkce vrací pouze jméno a ID všech úloh v kolekci exercises.

Závěr

Podařilo se vyvinout jednoduchý systém pro výuku programování v JS založený na jednoduchých cvičeních a automatizovaných testech. Systém je nasazen ve webovém prostředí na adrese https://ulohy-js.vercel.app/. Je zabezpečený proti běžným typům útoků a veřejně dostupný.

Oproti původnímu zadání práce se po domluvě s vedoucím přiklonilo k vytvoření systému pro ukládání a vykonávání úloh, nežli ke sbírce soustředící se jenom na vytváření různých cvičení.

Zdroje

[1] „jsfiddle,“ [Online]. Available: https://jsfiddle.net/. [Přístup získán 12 1 2021].

[2] „react.org,“ [Online]. Available: https://reactjs.org/. [Přístup získán 7 2 2021].

[3] „firebase.google.com,“ [Online]. Available: https://firebase.google.com/docs. [Přístup získán 2 2021].

[4] „firebase.google.com,“ [Online]. Available: https://firebase.google.com/docs/auth. [Přístup získán 17 1 2017].

[5] „firebase.google.com,“ 17 1 2021. [Online]. Available: https://firebase.google.com/docs/firestore.

[6] „codePen,“ [Online]. Available: https://codepen.io/quezo/post/stopping-infinite-loops. [Přístup získán 12 2 2021].

[7] „ReactJs,“ [Online]. Available: https://reactjs.org/docs/context.html. [Přístup získán 23 2 2020].

Obrázky