Střední průmyslová škola a Vyšší odborná škola, Liberec, příspěvková organizace

GRAPH-LIB

Ročníková práce

Autor	**Jakub Havlas**
Obor	**Informační technologie**
Vedoucí práce	**Mgr. Jan Šimůnek**
Školní rok	**2024/2025**
Počet stran	**25**
Počet slov	**4323**

Anotace

Moje práce se zabývá vytvářením grafů matematických funkcí. Vychází z aplikace GeoGebra, kterou jsem se inspiroval k tomuto projektu. Cílem práce je vytvořit zjednodušenou verzi Geogebry jako knihovnu v Reactu a následně ji demonstrovat na prototypové aplikaci.

Summary

My thesis focuses on creating graphs of mathematical functions. It is based on the GeoGebra application, which I inspired with. The goal of the project is to develop a simplified version of Geogebra as library in React and then demonstrate it through a prototype application.

Čestné prohlášení

Prohlašuji, že jsem předkládanou maturitní práci vypracoval sám a uvedl jsem veškerou použitou literaturu a bibliografické citace.

V Liberci dne 16.05.2025 .....

Jakub Havlas

Obsah

Úvod

Tuto práci jsem si vybral, protože mě baví matematika a chtěl bych se jí v budoucnu zabývat. Myslím, že zpracování matematických funkcí mě posune jak v programování, tak v matematický dovednostech.

1 Úvod do technologií

Svou aplikace jsem se rozhodl pojmout jako plně frontendovou aplikaci. To znamená, že nebudu potřebovat žádný server. Jako framework jsem si vybral React, protože je to jediná technologie, kterou umím a chtěl bych v ní prohloubit své dovednosti.

Dále jsem musel vymyslet, jak budu vlastně vykreslovat matematické funkce. Měl jsem na výběr mezi Canvas a SVG. Myslel jsem si, že SVG bude na 100% lepší, a ono je taky jednodušší na použití, ale teď vidím, že použití Canvas by mělo také své výhody.

2 Úvod do SVG[1]

Jako způsob vykreslování grafů jsem si zvolil programatické vykreslování ve formátu SVG. SVG se pro tuto práci hodí především proto, že nejde o rastrový, ale o vektorový formát. To znamená, že namísto práce s pixely a barvami mohu jednoduše definovat body, vektory a křivky, a tento formát je automaticky převede na pixely potřebné pro zobrazení na monitoru. Výhodou tohoto přístupu je také to, že se nemusím starat o ostrost obrazu – jak již bylo zmíněno, formát si ji dopočítá automaticky. Další výhodou formátu SVG je, že je to nativní formát a chápe jej každý prohlížeč. To znamená, že si nemusím psát žádnou funkci na zobrazování samotného grafu.

S tímto formátem jsem vymyslel dva způsoby tvorby práce. Obecný a Beziérový.

2.1 Obecný způsob

Prvním zvoleným přístupem je tzv. obecný způsob vykreslování. Tento postup spočívá ve vytvoření univerzální funkce, která na základě zadané matematické funkce vypočítá předem definovaný počet bodů v rámci zobrazeného prostoru (viewportu). Hlavní výhodou tohoto řešení je jeho univerzálnost – pro celou aplikaci postačí jedna vykreslovací funkce. Na druhé straně tento přístup klade vyšší nároky na správu detailnosti zobrazení. Tento způsob také vyžaduje, aby jedna funkce dokázala vyřešit jakoukoli funkci. Na druhou stranu je tento způsob je výpočetně náročnější.

2.2 Bézierovy křivky

Druhý přístup, který jsem zvažoval, využívá tzv. Bézierovy křivky. Tyto křivky se zapisují například následujícím způsobem:

```
d="M 100 350 Q 225 50 350 350"
```

V uvedeném příkladu si můžeme všimnout písmene **Q**, které označuje kvadratickou Bézierovu křivku (*quadratic Bézier curve*). Za tímto znakem následují dvě sady souřadnic. Poslední bod [350,350] představuje koncový bod, tedy místo, kde daná trajektorie končí. První bod [225,50] je tzv. **řídicí bod** (*control point*), který ovlivňuje zakřivení křivky.

Princip fungování Bézierových křivek spočívá v tom, že místo přímého spojení dvou bodů je výsledná dráha modifikována právě pomocí řídicího bodu. V případě kvadratické Bézierovy křivky je tvar určen jedním řídicím bodem, což umožňuje vykreslit

[1] SVG – scalable vektor graphics

hladkou a plynule zakřivenou trajektorii. Tato křivka nezačíná ani nekončí v řídicím bodě, ten pouze určuje její průběh.

Tento způsob zápisu je velmi užitečný při vykreslování hladkých oblouků, zakřivených linií nebo dekorativních prvků ve vektorové grafice. Díky přesné kontrole nad zakřivením lze pomocí Bézierových křivek vytvářet vizuálně atraktivní a plynulé tvary, které se hojně využívají jak v SVG grafice, tak v animačních nástrojích.

Tento způsob by byl o mnoho méně náročnější na systém, protože myslím, že formát SVG bude mít o mnoho lépe zvládnutou vykreslovací matematiku, než bude ta má. Na druhé straně to má i své nevýhody. Za prvé bych se musel naučit počítat Kontrolní bod z křivky, což jsem zjistil není úplně jednoduchý úkol. Za druhé, jelikož každá funkce vyžaduje jiný počet kontrolní bodů by to znamenalo, že každá funkce by měla vlastní komponentu v Reactu. Takže by bylo složité rozhodovat kam patří jaká funkce.

2.3 Finální návrh

Po konzultaci s vedoucím práce jsme se rozhodli, že nejvhodnější bude začít s obecným způsobem vykreslování. Toto rozhodnutí považuji za správné – jednak implementace univerzální funkce schopna řešit veškeré funkce je mnohem zajímavější z hlediska programování. Za druhé by byla komponenta ovladače pro výběr matematické funkce zbytečně složitá například kvůli funkci:

$$y = \sqrt{\frac{x+1}{x-1}}$$

Ve výše uvedeném příkladu by bylo velice těžké vybrat o jakou funkci se jedná.

3 Struktura knihovny

Na začátku práce jsem si nebyl jistý, jakým způsobem knihovnu koncipovat – zda by měla poskytovat pouze funkci pro vytváření křivek, nebo rovnou celé grafy. Nakonec jsem se rozhodl zahrnout všechny možnosti.

Základní komponenta je tedy LibraryController. Tato komponenta je vlastně taková řídící jednotka celé knihovny. Další důležitá komponenta je General. Účel této komponenty bylo vytvořit graf každé funkce. Dále velice důležitá funkce je Evaluator, tato komponenta měla za úkol vypočítat každý výraz. A poslední důležitou Komponentou je Picker. Dále jsou už jen menší užitečné funkce, hooky nebo komponenty.

3.1 Library Controller

Jak jsem již řekl Komponenta Library Controller představuje „mozek“ celé knihovny, jenž zajišťuje správné vykreslení grafů podle zadaných parametrů.

Tato komponenta je dle mého názoru nejzajímavější z celé knihovny a doporučuji její využití, protože udělá všechno za vás. Jediný povinný parametr je poly typu „reqs“ kterýž obsahuje dvě řetězcové proměnné – expression čili výraz a color, barva.

Další parametry jsou už čistě nepovinné. Jelikož píši knihovnu, pro kterou nevím, k čemu by ji mohl budoucí uživatel užít, přidal jsem raději spoustu parametrů ovlivňují chování. Například, jestli budete schopni s grafem hýbat, výchozí zobrazovací pole, minimální a maximální šířka, zobrazování mřížky, souřadnic, a nakonec jestli je možné stáhnout graf jako SVG.

```
Type reqs = {
Expression: string;
Color?: string;
}
```

Tato komponenta tedy první interpretuje výrazy do způsoby, se kterým s ním umí pracovat má aplikace. To má za odpovědnost ParseExpression. Dále jsou všechny výrazy poslány do komponenty ProcessInput, která validuje každý výraz a vyřeší proměnné. Posledním krokem tohoto procesu je zaslat funkce do komponenty Picker.

Pak tato funkce, jak jsem již řekl, spravuje spouštění dalších funkci jako generate grid, zoom, move, touch. Také využívá háky jako useDebounce, který je z hlediska knihovny React zajímavý pro můj obor.

3.1.1 UseDebounce

Custom hook useDebounce slouží k odkládání spuštění funkce. Toto využití se až skvěle hodí jako řešení pro optimalizaci v mé aplikaci. Potýkal jsem se s optimalizační problémy spojené s tím, že když uživat chtěl zoomnout, tak se za každý, byť jen maličký pohyv re-renderovala celá komponenta.

3.1.2 GenerateGrid

GenerateGrid je krátká funkce napsanou umělou inteligencí, jejíž cílem bylo jen vypočítat tloušťku čar, fontu a křivek pomocí zadaného šířky zobrazovaného pole.

3.1.3 Zoom, Move, Drag

Tyto pohybové události byli vytvořené umělou inteligencí. Měli za úkol dovolit uživateli kompletní kontrolu nad ovládáním grafu. Jsou volitelné, takže se dají i vypnout a tím vytvořit statická souřadnicový systém.

3.1.4 Parse Expression

Funkce ParseExpression je zodpovědná za rozdělení zadaného výrazu do podoby, se kterou mohou efektivně pracovat ostatní části systému. Nejprve je celý výraz rozdělen na jednotlivé znaky a odstraněny všechny mezery. Následně probíhá první průchod polem znaků, během kterého se vyhledávají sousedící znaky tvořící názvy matematických funkcí, například sin, cos nebo tan. Tyto sekvence jsou sloučeny do jednoho prvku pole. Ve druhé fázi se obdobným způsobem spojují vícemístná čísla, včetně desetinných čísel. Třetí průchod je věnován detekci záporných čísel, tedy situací, kdy se vyskytuje sekvence tvořená operátorem – a číslem. V takovém případě se znaménko a číslo spojí do jednoho prvku. V posledním průchodu funkce řeší implicitní násobení, kdy například zápis „2(“ je převeden na „2*(“. Díky těmto krokům je výsledkem funkce pole prvků připravené k dalšímu zpracování.

3.1.5 Process Input

Funkce ProcessInput zajišťuje finální kontrolu vstupních dat a jejich přípravu pro výpočet. Na rozdíl od předchozích validací tato funkce zasahuje hlouběji do obsahu výrazu. V první řadě ověřuje, zda všechny použité proměnné byly předem inicializovány, a tedy mají přiřazenou konkrétní hodnotu. Pokud tomu tak není, funkce na tuto skutečnost upozorní. Dále je vstup rozdělen na jednotlivé části, konkrétně na výrazy a případné inicializace proměnných. V závěrečném kroku funkce projde všechny výrazy a všechny výskyty proměnných nahradí jejich příslušnými hodnotami. Výsledkem je tedy

vstup připravený pro samotné vyhodnocení, bez jakýchkoliv nesrovnalostí nebo nedefinovaných prvků.

3.2 General

Komponenta General je druhou nejdůležitější částí celé práce. Jestli je LibraryController považován za mozek, pak musí být General považován za srdce. Tato funkce byla asi nesložitější část práce a trvalo mi nejdéle ji vymyslet, napsat a ladit. Problém vždycky je, že tato komponenta je výpočetně náročná a vždy vypočítá aspoň 10000 bodů. V průběhu vývoje tato komponenta začala využívat částečný výpočet grafu, kdy vypočítal jen části, které nebyly v předchozím zobrazovacím poli vidět, ale k tomu se dostaneme později.

3.2.1 useMemo

Hook useMemo je nativní hák v reactu. Dovoluje tzn. „Memoizovat“ výsledek funkce, dále definujeme závislosti jako druhý argument tohoto háku. Takže poprvé se funkce spustí, ale pokud je v budoucnu potřeba re-render a zadané závislosti se nezměnili, tak hoko vrátí „Memoizovanou“ hodnotu. Tento háček se skvěle hodil do kombinace s částečným výpočtem, protože ušetřil spoustu zbytečných výpočtů.

3.3 Evaluator

K vyhodnocování matematických výrazů, které mohou obsahovat proměnnou x, funkce (například sin, log) i matematické konstanty (π, e, aj.), je použita funkce evaluator. Tato funkce je napsaná tak, že si poradí s jakýmkoli matematickým výrazem, pokud je na vstupu ve validním tvaru. Tato funkce umí řešit, základní operace (+, - ,…) dále matematické funkce jako sin, tan, pak ještě závorky a to i absolutní hodnoty. Jako poslední funkce evaluatoru je, že dokáže řešit výrazy se známými konstantami, jako jsou π, e. Co ale Evaluator neumí, je jakékoli zjednodušování výrazů. To znamená, že vypočítá každý výraz, tak jak je napsaný do vstupu. Toto může taky vést k budoucím optimalizačním problémům.
Achillova pata této funkce je, že je drahá. Tato funkce obsahuje několik vnořených forloopů a ještě rekursivní volání v závorkách. Proto bylo cílem většiny optimalizací spouštět tuto funkci co nejméně.

3.4 Picker

Funkci picker momentálně funguje jako mezi komponenta, které je mezi LibraryController a General. Momentálně nemá velkou funkcionalitu, ale v budoucnu by

se projevila jako důležitou částí knihovny, kdybych ji chtěl obohacovat o další funkcionalitu, například kuželosečky. Pro ty není Evaluator ani General připraven, a proto by se musela napsat kompletně nová sada komponent. Dále si myslím, že by v tomto případě bylo lepší využívat Bézierovy křivky na místo obecnému řešení, ale to je čistě teoretická část.

4 Implementace

V této kapitole jsou popsány zajímavé technické prvky systému a obecně prvky, na kterém jsem hrdý a myslím, že by si měli zasloužit vlastní část.

4.1 Asymptote-detection tool

Tato implementaci mi zabrala vymyslet nejdelší dobu. V normálním způsobem se zde dají využít derivace, ale to není úkolem mé práce, takže jsem musel vymyslet jiný způsob. Tento způsob funguje vlastně docela jednoduše. Komponenta porovnává současný a poslední bod. Když je současný bod nad hranicí zobrazeného prostoru a poslední také, ale na druhé straně, aplikace jej prohlásí za asymptotu.

Toto řešení ovšem přináší dva nové problémy:

1. Co když bude funkce tak strmá, že mezi dvěma body přeskočí celý graf? Aplikace se rozbije.
2. A co když se další bod náhodou neocitne na opačné straně? Aplikace se rozbije.

První problém mě zas tolik netrápil, protože na to, aby taková situace nastala by musel uživatel nastavit polynom snad na $1000x^{1000}$.

Větší paseku mi nadělal druhý problém. Ono totiž jsem přijmul, že vždy budu pracovat s pravděpodobností, ale musel jsem přizpůsobit šance v můj prospěch. Tento problém jsem se rozhodl vyřešit SmartStepem o kterém si povíme v další kapitole. Protože jiné řešení mě prostě nenapadalo.

Tento problém jsem se rozhodl vyřešit SmartStepem. O kterém si povíme v další kapitolce.

4.1.1 Asymptoty

Největší výzvu, kterou jsem zatím řešil byli asymptoty. Tato programatická výzva se možná nezdá jako takový problém. Ale vlastně přinesla víc komplikací, než si jsem ochotný připustit. První jsem musel vymyslet, jak takovou asymptotu budu detekovat. Co to vlastně asymptota je? Asymptota je přímka, ke které se funkce nekonečně přibližuje ale je v ní nedefinovaná. Prvním krokem bylo, že jsem musel přidat podmínku pro žádné řešení. Asymptoty se například vyskytují v lineární lomené a v tangentové funkci. Ve všech případech se funkce blíží asymptotě jak pro záporné hodnoty y, tak pro kladné.

Můj nápad na řešení asymptot byl tedy jednoduchý. Když funkce vyleze z vertikálního zobrazeného prostoru. A na dalším bodě je další bod také za hranicí zobrazeného prostoru, ale na druhé straně. Je tato část prohlášena za asymptotu.

Toto řešení, byť jednoduché přinášelo spoustu problému a podmínek které bylo nutné ošetřit.

Za prvé bylo potřeba změnit datovou strukturu pathArray z coords[] na coords[][]. To zapříčinilo, že asymptoty byli vykreslované správně. ¨

Další a největší problém je náhoda. Jak jste jistě mohli odhadnout, tak můj algoritmus na hledání asymptot poměrně závisí na náhodu. Na konci vždy se budou porovnávat body, jestli jsou nad a pod zobrazeným prostorem. To zapříčinilo, že jediné, jak hledat efektivněji asymptoty bylo otočit šance víc v můj prospěch.

4.1.2 Intervaly

Na rozdíl od asymptot intervaly fungují trochu jednodušeji. Můj evaluator vrátí undefined „NaN“ protože se třeba snaží dělit nulou. Mám pak podmínku, která to chytí a přeskočí na další bod, a tak to funguje dál dokud se funkce nedostane na bod který má zase nějakou reálnou hodnotu. Nyní moje aplikace dokáže řešit funkce jako funkci níže které mají určitý interval při kterém tato funkce nemá reálné řešení.

$$y = \sqrt{\frac{x+1}{x-1}}$$

4.2 SmartStep

Při prvním hledáním optimalizací mě napadlo, že bych se měl snažit co nejvíce zmenšit počet vypočítaných bodů. Ale zmenšit počet bodů by vedlo k menšímu detailu. A tím i pro menší šanci nálezu asymptoty. Pro tento problém jsem vyřešil řešení označené jako „SmartStep“. Smartstep byl založen na myšlence, že bych mohl dokázat předpovídat, jak moc se funkce bude měnit. Kdybych to dovedl, tak bych pro hodně měnící se části zvětšil detail a pro málo měnící detail zmenšil.

Toto řešení ale také mělo své problémy. První bylo potřeba zjistit, jak spočítat větší změny hodnoty funkce? Za prvé jsem si řekl, že mě zajímá pouze y, takže potřebu najít jenom části, kdy y hodně klesá nebo stoupá. K tomuto úkolu vypadá, že se hodí jednoduše spočítat delta y, a to porovnávat s určitým prahem. V tomto řešení jsem viděl problém, že by neřešilo lineární funkce. Kdyby totiž někdo vložil y = 100x + 1 do vstupu, tak by se to pravděpodobně dostalo do prahu. A tím by byli lineární funkce zbytečně výpočetně náročné.

Útěchu jsem našel ve vektorové matematice. Ta mi totiž dovoluje řešit relativní úhly, a to se skvěle hodilo pro hledání křivek. Ale jak můžete předpokládat, tak ani toto řešení taky není ideální. Je patrně výpočetně náročnější a také vyžadovalo složitější implementaci. Déle se ukázalo jako problém, že relativní úhel také nestačí k rozhodnutí u předpokladu. Proto jsem musel zkombinovat obě řešení. A tím se celá implementace stala ještě výpočetně náročnější. Stále si myslím, že kdybych tomu věnoval dost času, tak by se z toho mohlo stát efektivní řešení.

Tou dobou jsem měl konzultaci s vedoucím práce a tam jsme se dohodli, že delta y bude stačit. A že nakonec nevadí, že budou lineární funkce náročnější.

4.3 CSS proměnné

Před publikování jsem začal přemýšlet, jak nastylovat mou knihovnu, tak aby byla co nejvíce modulární. Po krátkém průzkumu jsem se rozhodl použít cestu přes CSS proměnné.

Tato cesta funguje tak, že veškeré styly zabalím do proměnných a ty si pak uživatel bude moct upravit podle svých potřeb. Z toho plyne, že si uživatel může vybrat jakou barvu budou mít pomocné čáry, text nebo i pozadí. Také jsem v této části přidal nativní podporu světlého a tmavého režimu.

4.4 ExpressionValidation

Funkce ExpressionValidation slouží jako jednoduchý a rychlý nástroj, který je dostupný přímo uživateli. Jejím cílem je umožnit ověření, zda je zadaný matematický výraz syntakticky v pořádku a zda splňuje základní pravidla pro povolené znaky a správné použití funkcí a závorek. Je však důležité zdůraznit, že tato funkce neprovádí kontrolu úplnosti výrazu z hlediska inicializace proměnných. Jinými slovy, výraz může projít touto kontrolou jako validní, i když obsahuje proměnné, kterým nebyla přiřazena žádná hodnota. Funkce tedy slouží především k předběžné validaci a odhalení základních chyb ve struktuře zápisu, nikoliv k finálnímu ověření připravenosti výrazu k výpočtu.

4.4.1 Validatechars

Funkce ValidateChars slouží k ověření, zda výraz obsahuje pouze povolené znaky. Mezi povolené znaky patří operátory, čísla, proměnné a matematické funkce, jako například sin, cos, tan a další. Cílem této funkce je zajistit, aby se ve výrazu nenacházel žádný nepovolený znak, který by mohl způsobit chybu při jeho zpracování. Pokud funkce nalezne znak, který není součástí předem definovaného seznamu povolených prvků, označí jej jako nevalidní a může vrátit seznam těchto nevalidních znaků.

4.4.2 ValidateSyntax

Úlohou funkce ValidateSyntax je kontrolovat správnost zápisu matematického výrazu z hlediska syntaxe. Tato funkce například kontroluje, zda po zápisu matematické funkce následuje otevřená závorka. Dále ověřuje, že se v celém výrazu vyskytuje sudý počet znaků pro absolutní hodnoty, tedy znaků |, které musí být vždy v páru. Současně hlídá správné párování jednoduchých závorek, tedy že každý otevřený znak „(“ má na odpovídajícím místě svůj uzavřený „)“ a že se tyto závorky na konci výrazu navzájem

vynulují. Cílem této kontroly je zabránit nesmyslným zápisům, které by vedly k chybě při vyhodnocování výrazu.

4.5 DowlandButton

DowlandButton je krátká komponenta, která dovoluje uživateli stáhnout si souřadnicový systém s grafem jako soubor. Její parametr je řetězec znaků jako elementId. Po stisknutí tlačítka, tato funkce zabalí graf a stáhne ho jako soubor.

4.6 Cache

Docela dlouho jsem si kladl otázku, kam mám ukládat data. Zde v těchto třech kapitolách je popsán můj postup. Na začátek bych měl říct, že touha ukládat data se objevila v mé mysli ve chvíli, když jsem se rozhodl implementovat částečný výpočet.

4.6.1 LocalStorage

Prvním nápadem bylo použít local nebo session storage. Myslel jsem si, že to bude jednoduché řešení. Nyní však vím, že to tak není. Nejhorší problém je ten, že maximální velikost Session a Local storage je 5 MB to vůbec nestačilo, protože jsem chtěl ukládat pole s tisícími elementů. Další problémem bylo, že do tohoto způsobu ukládání lze ukládat pouze řetězec znaků. Na závěr jsem došel k závěru, že local a session storage jsou pouze k ukládání k uchovávání dočasných nebo trvalých uživatelských nastavení a dat mezi stránkami.

4.6.2 IndexedDatabase

Jako další technologií v mém seznamu byla IndexedDatabase. Ta funguje jako databáze ale na straně klienta a dovoluje ukládat složitější data a funguje plně asynchronně.

Kdybych se rozhodl jít cestou Indexed database, musel bych předělat své komponenty, aby dokázali řešit asynchronního programování. Myslím si, že to by způsobilo spoustu problémů, mnohem komplexnější chyby jen kvůli ukládání dat. V té době jsem si poprosil o konzultaci s panem Stehlíkem, který mě utvrdil, že toto cesta není.

4.6.3 Proměnná

Jak jsem již zmínil, v té době jsem měl konzultaci s panem Stehlíkem a tam jsem se rozhodl jít pro následující řešení. Doporučil mi použít useMemo a dát data prostě do proměnné. useMemo už jsem již popsal, takže nyní popíši jen proměnou AppData.

Vytvořil jsem proměnou, pole FunctionData.

```
export type FunctionData = {
  id: number;
  color: string;
  expression: string[];
  pathArray: coords[][];
};
```

Tato proměnná se stala jakoby mou tabulkou, ale musel jsem využít problém. Jak ji pak nadále exportuji pro uživatele, aby si mohli kontrolovat uložené funkce. Tento problém jsem vyřešil tím, že jsem k této proměnné napsal funkce CRUD a ty pak exportoval. Tím jsem zajistil, že uživatel bude mít naprostou kontrolu nad funkcemi uloženými v cache.

5 Nasazování a publikace

Finálním krokem mé práce bylo exportováním knihovny a publikováním ji na NPM.

5.1 Knihovna

Začal jsem tím, že jsem musel předělat strukturu knihovny pro snadnější export. Pro to jsem si založil nový projekt a github repositář. Zde každá komponenta měla vlastní složku. Dále zde byla přidána složka types, styles a utils. Poté byl vytvořen index.ts v tomto souboru jsou importovány a exportovány všechny komponenty nebo funkce, které jsem si vybral.

5.1.1 Bundling

JavaScriptové knihovny se nejprve musí takzvaně bundling, tedy sloučit do jednoho nebo několika výstupních souborů. Tento proces usnadňuje načítání skriptů vprohlížeči, zlepšuje výkon aplikace a umožňuje efektivnější správu závislostí. Během bundling dochází mimo jiné i k optimalizacím, jako je minifikace (zmenšení velikosti souborů odstraněním nepotřebných znaků) nebo tzv. tree shaking, který odstraní nepoužívaný kód. Mezi nejběžnější nástroje pro bundling patří Webpack, Vite nebo esbuild.

5.1.1.1 Rollup x webpack

Pro svůj projekt jsem si zvolil nástroj Rollup místo Webpacku, protože je jednodušší na konfiguraci a optimalizovaný pro tvorbu knihoven a balíčků s menším výsledným svazkem. Rollup navíc efektivně pracuje s ES moduly a generuje přehlednější a rychlejší kód pro prohlížeče.

5.1.2 Npm

Posledním krokem byla publikace mé knihovny na registr NPM, aby bylo stahování mé knihovny jednoduché. První jsem musel nastavit verzi na 1.0.0 tomu se říká SemVer, je to konvence, která se dá nejjednodušeji popsat jako: “MAJOR.MINOR.PATCH“. První číslo je důležitá aktualizace to znamená, že to není zpětně kompatibilní. Druhé číslo je menší aktualizace, ale je stále zpětně kompatibilní a třetí číslo je záplata. Pak bylo zapotřebí jen vymyslet jméno – graph-lib a bylo vše hotovo. Nyní si kdokoli může stáhnout moji knihovnu přes použití příkazu:

```
npm i @jakub-havlas/graph-lib@latest
```

6 Prototypová aplikace

Jako poslední část mojí práce bylo odprezentovat mou knihovnu na prototypové aplikaci. Tato aplikace obsahovala jen jednu komponentu, funkci a css soubor. Jelikož to také nebylo mým hlavním cílem, tak jsem se to snažil spíše osekat.

6.1 UserInput

Jediná komponenta aplikace, která zajišťuje vhodné plnění funkcí. Na mobilu se z ní stane tlačítko s vysouvacím překrytím jako řešení mobilního rozhraní. Sice to není ideální řešení, ale nepřišel jsem na nic lepšího. Tato komponenta využívá funkci z knihovny validateExpression aby uživateli podtrhla textové pole, které bylo špatně zapsáno. Dále tato komponenta využívá MathJax což je komponenta, která umí vykreslovat výrazy matematickým zápisem. K tomu byla ještě potřeba funkce ConverToMathJax, která byla napsaná umělou inteligencí, která z konvertuje řetězec znaků do Latexového zápisu MathJaxu. Jako poslední vlastností komponenty UserInput je, že podporuje klávesovou zkratku ctrl+z.

Závěr

Cílem této práce bylo vytvořit odlehčenou knihovnu pro tvorbu grafů a následně demonstrovat její možnosti na aplikaci inspirované Geogebrou.

Myslím si, že porovnávat mou knihovnu přímo s Geogebrou není příliš smysluplné. Jedinou výraznou výhodou mé knihovny je to, že je zcela open-source a velmi jednoduchá na použití. Spíše by bylo vhodné porovnávat ji s jinými knihovnami se stejným zaměřením. V tomto ohledu však mé řešení zaostává, především proto, že ostatní využívají pokročilejší matematické metody.

Přesto svou práci považuji za úspěšnou a rád bych se podobným tématům věnoval i nadále. Možná na ni navážu i ve své maturitní práci.

Rád bych se také zmínil o možnostech dalšího rozvoje tohoto projektu. Knihovna by se dala rozšířit o nové funkce a zlepšit její funkcionalita. Uvažovat lze i o přidání zcela nové oblasti matematiky, například planimetrie nebo kuželoseček. V neposlední řadě by stálo za to zaměřit se na optimalizaci celého řešení.

Na této práci jsem strávil přibližně 70 hodin. Přehled rozložení času je uveden v následující tabulce.

Celkem	Programování	Plánování	konzultace	Učení	dokumentace	Ladění a Nasazování
70,05 h	37,08 h	2,67 h	2,58 h	5,08 h	15 h	7,67 h

Seznam zkratek a odborných výrazů

SVG
Scalable vector graphics – formát obrázků využívajíc vektory namísto pixelů

Rollup
modulární bundler zaměřený na malé a optimalizované balíčky.

NPM
Node Package Manager – nástroj pro správu a instalaci JavaScript knihoven a závislostí v projektech.

MathJax
JavaScript knihovna pro zobrazování matematických výrazů na webu.

Tree shaking
technika odstraňování nepoužívaného kódu při buildování aplikace.

Bundling
proces spojování více zdrojových souborů do jednoho nebo několika bundle souborů.

Webpack
populární JavaScript bundler s podporou pluginů a loaderů.

Vite
moderní nástroj pro rychlý vývoj a buildování frontendových aplikací.

Esbuild
extrémně rychlý JavaScript bundler a minifikátor napsaný v Go.

SemVer
Semantic Versioning – konvence verzování ve formátu MAJOR.MINOR.PATCH podle typu změny.

Použité zdroje

1. **Sun, Y.** How to build a component library with React and TypeScript. *logrocket.* [Online] 2024. [Citace: 23. 4 2025.] https://blog.logrocket.com/how-to-build-component-library-react-typescript/.
2. **Harris, Rich.** Rollup Introduction. *Rollupjs.* [Online] Rollup Contributors. [Citace: 23. 4 2025.] https://dev.to/alexeagleson/how-to-create-and-publish-a-react-component-library-2oe.
3. **Eagleson, Alex.** How to Create and Publish a React Component Library. *dev.* [Online] DEV Community, 5. 12 2022. [Citace: 23. 4 2025.] https://dev.to/alexeagleson/how-to-create-and-publish-a-react-component-library-2oe.
4. **W3Schools.** SVG Path. *W3Schools.* [Online] [Citace: 16. 4 2025.] https://www.w3schools.com/graphics/svg_path.asp.
5. **Mozilla Foundation.** Paths – SVG from scratch. *MDN Web Docs.* [Online] Mozilla Foundation. [Citace: 16. 04 2025.] https://developer.mozilla.org/en-US/docs/Web/SVG/Tutorials/SVG_from_scratch/Paths.
6. **REICHL, Jaroslav.** Funkce – přehled učiva. *Matematika – Jaroslav Reichl.* [Online] 2013. [Citace: 16. 4 2025.] http://www.jreichl.com/matematika/vyuka/texty/fce_prehled.pdf.
7. **MDN Web Docs.** IndexedDB API. *MDN Web Docs.* [Online] Mozilla. [Citace: 17. 4 2025.] https://developer.mozilla.org/en-US/docs/Web/API/IndexedDB_API.
8. **React.** useMemo - React. *React.* [Online] Meta Platforms. Inc. [Citace: 17. 4 2025.] https://react.dev/reference/react/useMemo.
9. **MathJax Consortium.** MathJax Documentation. *MathJax.* [Online] MathJax Consortium, 2024. [Citace: 8. 5 2025.] https://docs.mathjax.org/.

A. Seznam přiložených souborů

Na přiloženém datovém nosiči se nacházejí následující soubory a složky:

- **RP2025-Havlas-Jakub-P3A-GeogebraLite.docx** – editovatelná verze dokumentace maturitní práce
- **RP2025-Havlas-Jakub-P3A-GeogebraLite.pdf** – tisknutelná verze dokumentace maturitní práce
- **Github aplikace –** veškeré kódy https://github.com/pslib-cz/RP2024-25_Havlas-Jakub_Geogebra-lite
- **Github knihovny – repositář** knihovny https://github.com/JakubHavlas/graph-lib
- **NPM page** – stránka na kterou byl výsledek publikován https://www.npmjs.com/package/@jakub-havlas/graph-lib
- **Přehled promptů** – v GitHub repositáři pod názvem „ChatGpt“
- **Nasazená aplikace** – https://pslib-cz.github.io/RP2024-25_Havlas-Jakub_Geogebra-lite/