Sem vložte zadání Vaší práce.

České vysoké učení technické v Praze
Fakulta informačních technologií
Katedra softwarového inženýrství

Bakalářská práce

Webová aplikace pro násobení smíšených polynomů

Herbert Waage

Vedoucí práce: Ing. Ivan Šimeček, Ph.D.

10. května 2015

Poděkování

Na tomto místě bych rád poděkoval vedoucímu práce Ing. Ivanu Šimečkovi, Ph.D. za jeho čas, ochotu a cenné rady poskytnuté při psaní této práce. Další poděkování patří mé rodině a přátelům, kteří mě podporovali během celého mého studia. V neposlední řadě bych chtěl poděkovat všem testerům, kteří mi pomohli výslednou aplikaci otestovat.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval(a) samostatně a že jsem uvedl(a) veškeré použité informační zdroje v souladu s Metodickým pokynem o etické přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 46 odst. 6 tohoto zákona tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou, a veškeré jejich dokumentace (dále souhrnně jen „Dílo“), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla, a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu), licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 10. května 2015 .....

České vysoké učení technické v Praze
Fakulta informačních technologií

Odkaz na tuto práci

Waage, Herbert. *Webová aplikace pro násobení smíšených polynomů.* Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2015.

Abstrakt

Práce se zabývá analýzou, návrhem a implementací webové aplikace pro násobení smíšených polynomů. Aplikace využívá triviální algoritmus, Strassenův algoritmus a algoritmus pro násobení řídkých smíšených polynomů ve formátu Compressed Row Storage. Tyto algoritmy jsou v práci také popsány po teoretické stránce. Výsledkem je aplikace, která dokáže vynásobit 2 smíšené polynomy o 2 neznámých pomocí zvoleného algoritmu a vykreslit grafy rychlostí různých výpočtů.

Klíčová slova smíšený polynom, násobení, paralelizace

Abstract

This thesis deals with analysis, design and implementation of a web application for multiplication of the mixed polynomials. The application uses trivial algorithm, Strassen's algorithm and algorithm for multiplication of sparse mixed polynomials in the Compressed Row Storage format. The work contains theoretical description of these algorithms. The result is a web application, which can multiply 2 bivariate polynomials using the selected algorithm and draw plots of speeds of different algorithms.

Keywords mixed polynomial, multiplication, parallelization

Obsah

Seznam obrázků

Seznam tabulek

Úvod

Motivace

Násobení smíšených polynomů je jedna ze základních operací nejen v matematice, ale také ve světě počítačových technologií. Postupem času se objevila řada algoritmů umožňujících jejich násobení rychleji než to dokáže klasický triviální algoritmus násobením každého prvku prvního polynomu s každým prvkem druhého polynomu. Tyto algoritmy používají také různé formáty uložení polynomů, což umožňuje zefektivnit i paměťovou náročnost daného algoritmu. Tato práce si klade za cíl vytvořit webovou aplikaci umožňující porovnávat algoritmy pro násobení smíšených polynomů o dvou neznámých.

Struktura práce

Práce je rozdělena do několika následujících kapitol:

- Kapitola 1 „**Teoretický základ**“ vysvětluje základní pojmy potřebné pro pochopení problematiky a samotné algoritmy pro násobení polynomů.
- Kapitola 2 „**Analýza**“ popisuje požadavky, které jsou kladeny na výslednou aplikaci. Dále je zde popis možných případů užití a doménový model zahrnující i stavové diagramy.
- V kapitole 3 „**Návrh**“ je konceptuální model databáze vycházející z doménového modelu, návrh grafického uživatelského rozhraní a popis použitých implementačních nástrojů. Je zde popsána i technologie OpenMP, která je použita při implementaci algoritmů pro násobení polynomů.
- Kapitola 4 „**Realizace a testování**“ se zabývá jednotlivými kroky implementace jak dané webové aplikace, tak i algoritmů pro násobení polynomů. Obsahuje i shrnutí výsledků uživatelského testování výsledné aplikace a testování korektnosti implementovaných algoritmů.

Kapitola 1

Teoretický základ

1.1 Základní pojmy

Součástí této bakalářské práce je násobení smíšených polynomů. Proto je potřeba se na začátku seznámit se základními pojmy související s touto problematikou. Při tvorbě této podkapitoly jsem čerpal z [16] a z [1].

1.1.1 Polynom

Definice *Polynom* je komplexní funkce komplexní proměnné, tedy $p : \mathbf{C} \to \mathbf{C}$, která má pro všechna $x \in \mathbf{C}$ funkční hodnotu $p(x)$ danou vzorcem

$$p(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n,$$

kde $a_0, \ldots, a_n$ jsou nějaká reálná nebo komplexní čísla, která nazýváme *koeficienty polynomu*.

Stupeň polynomu Nechť polynom p má koeficienty $a_0, a_1, \ldots, a_n$. *Stupeň polynomu* je největší index k takový, že $a_k \neq 0$. Pokud jsou všechny koeficienty nulové (nulový polynom), prohlásíme, že stupeň je roven -1.

Součin polynomů *Součin polynomů* p a q je funkce u dána předpisem $u(x) = p(x) * q(x)$ pro všechna $x \in \mathbf{C}$. Součin polynomů p a q značíme pq.

1.1.2 Smíšený polynom

Tato práce se zabývá násobením smíšených polynomů o pouze dvou neznámých, proto se budou následující definice zaměřovat pouze na tyto smíšené polynomy.

Definice *Smíšený polynom* je komplexní funkce komplexních proměnných (tato práce se zabývá dvěma proměnnými), tedy $p : \mathbf{C} \to \mathbf{C}$, která má pro všechna $x \in \mathbf{C}$ a $y \in \mathbf{C}$ funkční hodnotu $p(x, y)$ danou vzorcem

$$\begin{array}{rccc} & a_{m,n}x^m y^n & + \cdots + & a_{m,0}x^m y^0 + \\ & a_{m-1,n}x^{m-1}y^n & + \cdots + & a_{m-1,0}x^{m-1}y^0 + \\ p(x,y) = & \vdots & \ddots & \vdots \\ & a_{1,n}x^1 y^n & + \cdots + & a_{1,0}x^1 y^0 + \\ & a_{0,n}x^0 y^n & + \cdots + & a_{0,0}x^0 y^0, \end{array}$$

kde $a_{0,0}, \ldots, a_{m,n}$ jsou nějaká reálná nebo komplexní čísla, která nazýváme *koeficienty* smíšeného polynomu. Pokud od této chvíle budu hovořit o smíšeném polynomu, budu tím myslet smíšený polynom o dvou neznámých.

Velikost smíšeného polynomu Velikost smíšeného polynomu je počet nenulových koeficientů v daném polynomu.

Součin smíšených polynomů *Součin smíšených polynomů* p a q je funkce u dána předpisem $u(x, y) = p(x, y) * q(x, y)$ pro všechna $x \in \mathbf{C}$ a $y \in \mathbf{C}$. Součin smíšených polynomů p a q značíme pq.

Matice smíšeného polynomu *Maticí* $\mathbf{A}$ *smíšeného polynomu* $p(x, y)$ rozumíme matici, kde sloupce reprezentují mocninu proměnné x a řádky reprezentují mocninu proměnné y. Hodnota v matici představuje hodnotu koeficientu smíšeného polynomu.

$$\mathbf{A} = \begin{pmatrix} a_{0,0}x^0y^0 & a_{1,0}x^1y^0 & \cdots & a_{n,0}x^n y^0 \\ a_{0,1}x^0y^1 & a_{1,1}x^1y^1 & \cdots & a_{n,1}x^n y^1 \\ \vdots & \vdots & \ddots & \vdots \\ a_{0,m}x^0y^m & a_{1,m}x^1y^m & \cdots & a_{n,m}x^n y^m \end{pmatrix}$$

Například matice $\mathbf{A}$ pro polynom $2x^2y^2 - 5xy + x + 3$ by vypadala následovně:

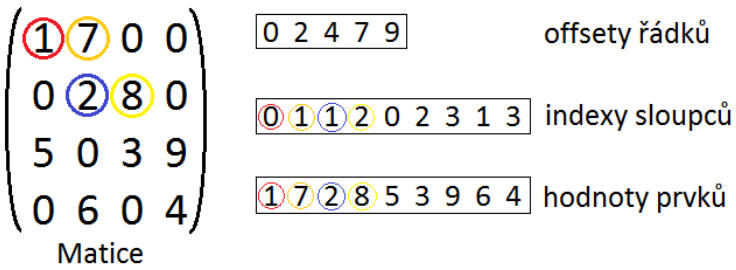
$$\mathbf{A} = \begin{pmatrix} 3 & 1 & 0 \\ 0 & -5 & 0 \\ 0 & 0 & 2 \end{pmatrix}$$

Od této chvíle pokud budu hovořit o polynomu, budu tím myslet jeho matici a značit ho budu velkými písmeny.

Počet nenulových koeficientů Při popisu algoritmů je důležité si stanovit pojem *počet nenulových koeficientů polynomu*, který budu značit nn_z.

1.1.3 Formát CRS

Formát CRS (**C**ompressed **R**ow **S**torage) někdy označovaný jako CSR (**C**ompressed **S**parse **R**ow) je formát pro efektivní uložení řídkých matic [18]. Na rozdíl od triviálního uložení všech prvků matice tento způsob ukládá pouze ty nenulové a to do následujících tří polí. V prvním poli jsou uloženy offsety řádků, v druhém indexy sloupců a ve třetím všechny nenulové prvky matice smíšeného polynomu přesně jak je demonstrováno na obrázku 1.1.



Obrázek 1.1: Vizualizace formátu CRS

1.2 Algoritmy pro násobení

Pro násobení polynomů existuje řada algoritmů. Některé z nich lze použít i pro násobení smíšených polynomů. Tato bakalářská práce se zabývá třemi algoritmy a to triviálním, Strassenovým a algoritmem pro násobení smíšených polynomů ve formátu CRS.

1.2.1 Triviální algoritmus

Základní algoritmus pro vynásobení 2 smíšených polynomů se nazývá triviální algoritmus. Je založen na násobení každého prvku polynomu **A** s každým prvkem polynomu **B** a to i včetně všech nulových prvků. Ukládání výsledku vynásobení probíhá přičtením k původní hodnotě ve výsledném polynomu na místě, které se určí jako součet řádků/sloupců původních polynomů, kde se řádky a sloupce číslují od 0.

Pro zjednodušení budu dále uvažovat, že matice polynomů jsou čtvercové a obě stejně veliké.

Jelikož se jedná o základní algoritmus pro násobení, tak asymptotická složitost je $O(n^2)$, kde n je počet koeficientů jednoho z původních polynomů. Algoritmus provede n^2 operací násobení a n^2 operací sčítání.

Pseudokód tohoto algoritmu můžeme vidět na ukázce algoritmu 1. Na řádku 1 vypočteme velikost matice výsledného polynomu a dále na řádcích 2-6 vyplníme celou výslednou matici nulami. Poté na řádcích 7-15 pomocí for cyklů projedeme všechny prvky polynomu **A** a každý tento prvek vynásobíme kaž-

dým prvkem polynomu **B**. Výsledek přičítáme k původní hodnotě v polynomu **C** na řádku 11.

Algoritmus 1 Pseudokód triviálního algoritmu

Require: A, B
Ensure: C

```
 1: matrixSize(C) ← matrixSize(A) + matrixSize(B) - 1
 2: for i ← 0 to (matrixSize(A) - 1) do
 3:     for j ← 0 to (matrixSize(A) - 1) do
 4:         C[i][j] ← 0
 5:     end for
 6: end for
 7: for i ← 0 to (matrixSize(A) - 1) do
 8:     for j ← 0 to (matrixSize(A) - 1) do
 9:         for m ← 0 to (matrixSize(B) - 1) do
10:             for n ← 0 to (matrixSize(B) - 1) do
11:                 C[i + m][j + n] ← C[i + m][j + n] + A[i][j] * B[m][n]
12:             end for
13:         end for
14:     end for
15: end for
16: return C
```

1.2.2 Algoritmus pro násobení řídkých smíšených polynomů s využitím CRS formátu

Další algoritmus pro násobení smíšených polynomů využívá formátu CRS popsaného v 1.1.3. Stejně jako triviální algoritmus i tento je založen na násobení každého prvku polynomu **A** s každým prvkem polynomu **B**. Jediný rozdíl je ve množství ukládaných dat v tom smyslu, že se neukládají nulové prvky. Výsledek vynásobení 2 prvků se ukládá přičtením k původní hodnotě výsledného polynomu na místo určené součtem řádků/sloupců původních polynomů.

Díky formátu CRS se sice nemusí ukládat nulové prvky, ale přibude informace o poloze prvků v matici polynomu. Právě proto se tento algoritmus hodí pro takzvané *řídké polynomy*, což jsou polynomy s větším počtem nulových koeficientů než těch nenulových [4].

Asymptotická složitost tohoto algoritmu je $O(nn_z(A) * nn_z(B))$, kde nn_z vyjadřuje počet nenulových koeficientů. Algoritmus provede $nn_z(A) * nn_z(B)$ operací násobení a $nn_z(A) * nn_z(B)$ operací sčítání.

Pseudokód tohoto algoritmu můžeme vidět na ukázce algoritmu 2. Vstupní polynomy dostáváme ve formátu CRS. Na řádku 1 vypočteme velikost matice výsledného polynomu a dále na řádcích 2-6 vyplníme celou výslednou matici nulami. Na řádcích 7-23 probíhá samotný algoritmus, kde iterujeme přes všechny prvky polynomu **A** a násobíme je s každým prvkem polynomu **B**.

Výsledek přičítáme k původní hodnotě ve výsledném polynomu na řádce 17. Některé proměnné jsou pojmenované zkratkami, kde „i“ znamená index, „r“ řádka a „c“ sloupec.

Algoritmus 2 Pseudokód algoritmu využívajícího CRS formátu

Require: CRS A, CRS B
Ensure: C

```
 1: matrixSize(C) ← matrixSize(A) + matrixSize(B) - 1
 2: for i ← 0 to (matrixSize(A) - 1) do
 3:     for j ← 0 to (matrixSize(A) - 1) do
 4:         C[i][j] ← 0
 5:     end for
 6: end for
 7: for i ← 0 to (count(ARowsOffsets) - 2) do
 8:     Ai ← ARowsOffsets[i] - 1
 9:     for j ← 0 to (count(ANumbersInRow) do
10:         Ac ← AColsIndices[i];
11:         Ar ← i;
12:         for m ← 0 to (count(BRowsOffsets) - 2) do
13:             Bi ← BRowsOffsets[m] - 1
14:             for n ← 0 to (count(BNumbersInRow) do
15:                 Bc ← BColsIndices[m]
16:                 Br ← m
17:                 C[Ar + Br][Ac + Bc]+ = AData[Ai] ∗ BData[Bi]
18:                 Bi ← Bi + 1
19:             end for
20:         end for
21:         Ai ← Ai + 1
22:     end for
23: end for
24: return C
```

1.2.3 Strassenův algoritmus

Strassenův algoritmus je jeden z algoritmů, který se používá pro násobení matic. Poprvé ho představil Volker Strassen v roce 1949 [9]. Princip využití násobení matic pro násobení smíšených polynomů je popsán v kapitole 1.3.

Tento algoritmus potřebuje na vstupu 2 matice stejné velikosti a tato velikost musí být 2^n, kde $n \in \mathbf{N}$. Tento specifický formát matic potřebujeme z toho důvodu, abychom mohli matice rozdělovat na 2 stejné části. Pokud jsou matice jiného formátu, musí se patřičně rozšířit a nové prvky nastavit na hodnotu 0. Strassenův algoritmus má asymptotickou složitost $O(n^{log_2 7})$.

Oproti klasickému triviálnímu algoritmu, který má asymptotickou složitost $O(n^{log_2 8})$, totiž provede o jedno násobení matic méně. Toho docílí tím, že pro-

vede více sčítání/odčítání matic, které je oproti násobení mnohem jednodušší.

Nyní tedy již předpokládáme, že matice jsou v požadovaném formátu. Klasický triviální algoritmus počítá výslednou matici **C** následovně:

$$\mathbf{C} = \mathbf{A} * \mathbf{B} \qquad \mathbf{A}, \mathbf{B}, \mathbf{C} \in R^{2^n x 2^n}$$

Všechny matice se rozdělí na 4 stejně velké podmatice:

$$\mathbf{A} = \begin{pmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \end{pmatrix}, \mathbf{B} = \begin{pmatrix} B_{1,1} & B_{1,2} \\ B_{2,1} & B_{2,2} \end{pmatrix}, \mathbf{C} = \begin{pmatrix} C_{1,1} & C_{1,2} \\ C_{2,1} & C_{2,2} \end{pmatrix}$$

Nyní lze vyjádřit prvky výsledné matice **C** následujícím způsobem:

$$\begin{aligned} C_{1,1} &= A_{1,1} * B_{1,1} + A_{1,2} * B_{2,1} \\ C_{1,2} &= A_{1,1} * B_{1,2} + A_{1,2} * B_{2,2} \\ C_{2,1} &= A_{2,1} * B_{1,1} + A_{2,2} * B_{2,1} \\ C_{2,2} &= A_{2,1} * B_{1,2} + A_{2,2} * B_{2,2} \end{aligned}$$

Ale i po této úpravě zůstává počet násobení stejný. Musí se provést 8 násobení a 4 sčítání. Nyní přichází zásadní část úvahy, kdy definujeme nové matice, které následně vyjádří prvky výsledné matice **C**:

$$\begin{aligned} M_1 &= (A_{1,1} + A_{2,2}) * (B_{1,1} + B_{2,2}) \\ M_2 &= (A_{2,1} + A_{2,2}) * B_{1,1} \\ M_3 &= A_{1,1} * (B_{1,2} - B_{2,2}) \\ M_4 &= A_{2,2} * (B_{2,1} - B_{1,1}) \\ M_5 &= (A_{1,1} + A_{2,1}) * B_{2,2} \\ M_6 &= (A_{2,1} - A_{1,1}) * (B_{1,1} + B_{1,2}) \\ M_7 &= (A_{1,2} - A_{2,2}) * (B_{2,1} + B_{2,2}) \end{aligned}$$

Pro vyjádření matic M_i jsme potřebovali pouze 7 operací násobení. Pomocí těchto matic nyní lze bez přidání dalšího násobení vyjádřit celou výslednou matici **C** následujícím způsobem:

$$\begin{aligned} C_{1,1} &= M_1 + M_4 - M_5 + M_7 \\ C_{1,2} &= M_3 + M_5 \\ C_{2,1} &= M_2 + M_4 \\ C_{2,2} &= M_1 - M_2 + M_3 + M_6 \end{aligned}$$

Na jednotlivá násobení matic lze použít opět Strassenův algoritmus dokud se z matic nestanou matice řádu 1. V praxi se to ale nevyplatí a Strassenův algoritmus používá pro menší matice klasický triviální algoritmus.

Pseudokód tohoto algoritmu můžeme vidět na ukázce algoritmu 3. Číslo x představuje dělící hranici mezi Strassenovým a triviálním algoritmem. Na vstupu jsou 2 matice **A** a **B**, u kterých už v pseudokódu předpokládáme správný tvar pro Strassenův algoritmus. Na řádce 1-3 je přepnutí na triviální algoritmus, pokud je velikost matic již malá. Na řádcích 4-7 konstruujeme patřičné podmatice a dále na řádcích 8-14 z těchto podmatic vytváříme matice M_i. Na řádcích 15-19 konstruujeme z vypočtených matic výslednou matici **C**.

Algoritmus 3 Pseudokód Strassenova algoritmu pro násobení matic

Require: A, B
Ensure: C
1: **if** matrixSize(A) = x **then**
2: **return** trivialMultiplication(A, B)
3: **end if**
4: $A_{1,1} \leftarrow leftTop(A), B_{1,1} \leftarrow leftTop(B)$
5: $A_{1,2} \leftarrow rightTop(A), B_{1,2} \leftarrow rightTop(B)$
6: $A_{2,1} \leftarrow leftBottom(A), B_{2,1} \leftarrow leftBottom(B)$
7: $A_{2,2} \leftarrow rightBottom(A), B_{2,2} \leftarrow rightBottom(B)$
8: $M_1 \leftarrow (A_{1,1} + A_{2,2}) * (B_{1,1} + B_{2,2})$
9: $M_2 \leftarrow (A_{2,1} + A_{2,2}) * B_{1,1}$
10: $M_3 \leftarrow A_{1,1} * (B_{1,2} - B_{2,2})$
11: $M_4 \leftarrow A_{2,2} * (B_{2,1} - B_{1,1})$
12: $M_5 \leftarrow (A_{1,1} + A_{2,1}) * B_{2,2}$
13: $M_6 \leftarrow (A_{2,1} - A_{1,1}) * (B_{1,1} + B_{1,2})$
14: $M_7 \leftarrow (A_{1,2} - A_{2,2}) * (B_{2,1} + B_{2,2})$
15: $C_{1,1} \leftarrow M_1 + M_4 - M_5 + M_7$
16: $C_{1,2} \leftarrow M_3 + M_5$
17: $C_{2,1} \leftarrow M_2 + M_4$
18: $C_{2,2} \leftarrow M_1 - M_2 + M_3 + M_6$
19: $C \leftarrow$construct$(C_{1,1}, C_{1,2}, C_{2,1}, C_{2,2})$
20: **return** C

1.3 Využití násobení matic při násobení smíšených polynomů

Existuje postup, jak využít maticového násobení při násobení smíšených polynomů. Díky tomu můžeme využít Strassenova algoritmu.

Matice vstupních polynomů **A** a **B** rozšíříme tak, aby měli stejnou velikost a byly čtvercové. Nově vzniklé prvky nastavíme na 0. Matici **A** necháme v původním stavu a matici **B** transformujeme tak, aby exponenty x a y byly v matici rozmístěny následujícím způsobem:

$$\mathbf{B}_t = \begin{pmatrix} x^n y^0 & x^n y^1 & \cdots & x^n y^{n-1} & x^n y^n \\ x^{n-1} y^0 & x^{n-1} y^1 & \cdots & x^{n-1} y^{n-1} & x^{n-1} y^n \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ x^1 y^0 & x^1 y^1 & \cdots & x^1 y^{n-1} & x^0 y^n \\ x^0 y^0 & x^0 y^1 & \cdots & x^0 y^{n-1} & x^0 y^n \end{pmatrix}$$

Nyní po vynásobení matice **A** a transformované matice $\mathbf{B}_t$ získáme všechny kombinace mocnin $x^{\frac{n-1}{2}}$ se všemi mocninami neznámé y výsledného polynomu, kde n je rozměr matice výsledného polynomu. Napříč maticí se mocniny y

rovnají, proto pro vypočtení výsledné hodnoty koeficientů musíme všechny stejné kombinace mocnin sečíst.

Následuje podstatná část algoritmu, kdy získáme všechny další kombinace mocnin. Pokud chceme dostat všechny kombinace mocniny $x^{\frac{n-1}{2}-1}$, tak z matice $\mathbf{A}$ odstraníme poslední sloupec a z matice $\mathbf{B}_t$ odstraníme první řádek. Tím zachováme možnost násobit matice a snížíme výsledné mocniny x o 1.

$$\mathbf{A} = \begin{pmatrix} x^0y^0 & x^1y^0 & \cdots & x^{n-1}y^0 \\ x^0y^1 & x^1y^1 & \cdots & x^{n-1}y^1 \\ \vdots & \vdots & \ddots & \vdots \\ x^0y^{n-1} & x^1y^{n-1} & \cdots & x^{n-1}y^{n-1} \\ x^0y^n & x^1y^n & \cdots & x^{n-1}y^n \end{pmatrix}$$

$$\mathbf{B}_t = \begin{pmatrix} x^{n-1}y^0 & x^{n-1}y^1 & \cdots & x^{n-1}y^{n-1} & x^{n-1}y^n \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ x^1y^0 & x^1y^1 & \cdots & x^1y^{n-1} & x^0y^n \\ x^0y^0 & x^0y^1 & \cdots & x^0y^{n-1} & x^0y^n \end{pmatrix}$$

Pokud naopak chceme získat všechny mocniny $x^{\frac{n-1}{2}+1}$, tak odstaníme z matice $\mathbf{A}$ první sloupec a z matice $\mathbf{B}_t$ poslední řádek.

$$\mathbf{A} = \begin{pmatrix} x^1y^0 & \cdots & x^{n-1}y^0 & x^ny^0 \\ x^1y^1 & \cdots & x^{n-1}y^1 & x^ny^1 \\ \vdots & \ddots & \vdots & \vdots \\ x^1y^{n-1} & \cdots & x^{n-1}y^{n-1} & x^ny^{n-1} \\ x^1y^n & \cdots & x^{n-1}y^n & x^ny^n \end{pmatrix}$$

$$\mathbf{B}_t = \begin{pmatrix} x^ny^0 & x^ny^1 & \cdots & x^ny^{n-1} & x^ny^n \\ x^{n-1}y^0 & x^{n-1}y^1 & \cdots & x^{n-1}y^{n-1} & x^{n-1}y^n \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ x^1y^0 & x^1y^1 & \cdots & x^1y^{n-1} & x^0y^n \end{pmatrix}$$

Postup odstraňování řádků/sloupců z matic $\mathbf{A}$ a $\mathbf{B}_t$ se opakuje dokud nedostaneme všechny kombinace všech mocnin x se všemi mocninami y.

Kapitola 2

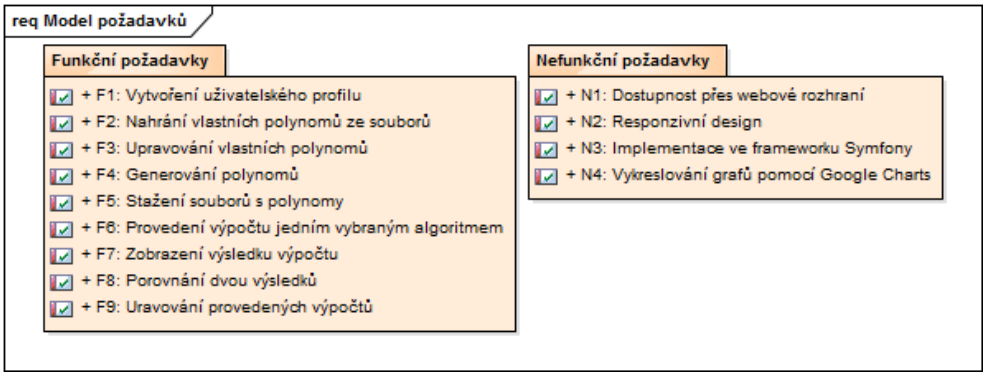
Analýza

Analýza je jeden ze základních stavebních kamenů při vývoji webové aplikace a potažmo i softwaru jako takového. Jedná se o popis toho, co má výsledná aplikace dělat. Kvalitně zpracovanou analýzou lze předejít chybám, které by vznikly při implementaci a bylo by náročné je odstranit po jejich vzniku.

Analýza podle [14] a [15] obsahuje seznam funkčních a nefunkčních požadavků na aplikaci, diagramy aktivit, diagramy případů užití a doménový model. Protože moje webová aplikace nebude nijak složitá, vynechám diagramy aktivit, které popisují tok událostí při konkrétním procesu.

2.1 Analýza požadavků

Analýza požadavků shrnuje funkční a nefunkční požadavky na výslednou aplikaci. Funkční požadavky popisují interakci aplikace s uživatelem zatímco nefunkční (obecné) požadavky určují nároky na implementaci, výkonnost a dostupnost aplikace.



Obrázek 2.1: Model požadavků

2.1.1 Splnění bodů zadání

Ještě než přistoupím k přesnému popisu jednotlivých požadavků, uvedu zde pro větší přehlednost, čím jsou splněny jednotlivé body zadání.

Co se týče popisu technologií, které se mají použít při implementaci, tak všechny tyto body splňují nefunkční požadavky. Splnění dalších bodů, které jsou v zadání očíslovány, demonstruje tabulka 2.1.

1. přihlášení
2. generování polynomů podle zadaných parametrů (stupeň, počet nenulových hodnot, ...)
3. uložení a načtení polynomů do/ze souborů ve specifikovaném formátu
4. volbu algoritmu pro násobení
5. zobrazení výsledku
6. seznam proběhnutých výpočtů přihlášeného uživatele
7. vykreslování grafů rychlostí různých výpočtů (po vybrání ze seznamu proběhnutých výpočtů) s využitím javascriptové knihovny Google Charts

Co se týče přihlášení, tak se jedná o autorizaci a autentizaci. Autentizace je proces ověření identity, který většinou předchází procesu autorizace, což je umožnění přístupu nebo nějaké akce daným uživatelem [12].

Tabulka 2.1: Tabulka splnění bodů zadání funkčními požadavky

	F1 2.2.1	F2 2.2.2	F3 2.2.3	F4 2.2.4	F5 2.2.5	F6 2.2.6	F7 2.2.7	F8 2.2.8	F9 2.2.9	splněno
1.	X									✓
2.				X	X					✓
3.		X								✓
4.						X				✓
5.							X			✓
6.									X	✓
7.								X		✓

Jak lze z tabulky vypozorovat, tak všechny body zadání jsou plněny jedním nebo i více funkčními požadavky. Funkční požadavek 3 implicitně neplní žádný bod zadání, ale byl přidán kvůli potřebě sledování uživatelových polynomů.

2.2 Funkční požadavky

2.2.1 F1 Vytvoření uživatelského profilu

Uživatel si bude moct vytvořit uživatelský profil, který bude použit pro přihlášení do aplikace. Profil bude obsahovat pouze uživatelské jméno a heslo. Pomocí tohoto profilu se bude moci do aplikace přihlásit.

2.2.2 F2 Nahrání vlastních polynomů ze souborů

Přihlášenému uživateli bude umožněno nahrát vlastní soubory s polynomy, které mohou být dále použity pro výpočty. Tyto soubory s polynomy budou muset mít přesně danou strukturu, jinak budou aplikací odmítnuty a po uživateli bude vyžadováno opravit daný soubor a znovu ho nahrát.

2.2.3 F3 Upravování vlastních polynomů

Jakmile si uživatel vytvoří nějaké polynomy, bude mu nabídnuto je smazat. Smazání bude umožněno v seznamu všech uživatelových polynomů. V detailu polynomu bude vykreslen graf poměru nulových a nenulových koeficientů polynomu, data polynomu a seznam provedených výpočtů s daným polynomem.

2.2.4 F4 Generování polynomů

Aplikace bude obsahovat generátor polynomů podle zadaných parametrů. Tyto parametry budou velikost polynomu a procentuální počet nulových koeficientů polynomu. Generátor nemusí být přesný v procentuálním počtu nulových koeficientů, ale měl by se blížit zadanému číslu. Počet nulových prvků bude přepočítán po vygenerování polynomu a uživatel dostane přesnou informaci o výsledném číslu.

2.2.5 F5 Stažení souborů s polynomy

Své polynomy bude uživatel schopen stáhnout ve formě textového souboru. Tato možnost bude dostupná ze seznamu všech uživatelových polynomů.

2.2.6 F6 Provedení výpočtu jedním vybraným algoritmem

Aplikace bude umožňovat provedení vynásobení dvou polynomů přihlášeného uživatele. Bude možnost pojmenování výpočtu, výběr algoritmu k provedení a počet vláken, na kterých výpočet poběží.

2.2.7 F7 Zobrazení výsledku výpočtu

Detail provedených výpočtů bude obsahovat informace o vstupních polynomech a výstupním polynomu, spojnicový graf zobrazující zrychlení na více

vláknech a koláčový graf, který bude představovat poměr mezi načtením vstupu, výpočtem a uložením výstupu. K detail výpočtu bude uživatel přistupovat přes seznam všech provedených výpočtů.

2.2.8 F8 Porovnání dvou výsledků

Dva provedené výpočty bude možno porovnat. Toto porovnání bude spočívat v porovnání rychlostí celkových výpočtů na různém počtu vláken a v poměrech načítání vstupu, výpočtu a ukládání výstupu.

2.2.9 F9 Upravování provedených výpočtů

U provedených výpočtů bude umožněno jejich smazání. Tato možnost bude přístupna ze seznamu všech uživatelem provedených výpočtů.

2.3 Nefunkční požadavky

2.3.1 N1 Dostupnost přes webové rozhraní

Výsledná aplikace bude dostupná přes webové rozhraní a prohlížeče Google Chrome (verze 40.0 a vyšší), Mozilla Firefox (verze 34.0 a vyšší) a Internet Explorer (IE 10 a vyšší).

2.3.2 N2 Responzivní design

Aplikace bude mít responzivní design. To znamená, že její vzhled se bude upravovat při zobrazování na různých zařízeních. Toho nebude docíleno pomocí javascriptu, ale pomocí *media queries*, flexibilního layoutu a flexibilních obrázků. Media queries je možnost definovat různý vzhled stránky pro různé zařízení/rozlišení [3].

2.3.3 N3 Implementace ve frameworku Symfony

Implementace aplikace bude provedena ve frameworku Symfony verze 2.6.1. Pro řídící logiku bude použit jazyk PHP, datový model bude zajišťovat objektově relační mapování pomocí Doctrine a pro uživatelské rozhraní bude použit šablonovací systém Twig.

2.3.4 N4 Vykreslování grafů pomocí Google Charts

Všechny dynamické grafy vyskytující se v aplikaci budou vykreslovány pomocí javascriptové knihovky Google Charts.

2.4 Případy užití

Modelování případů užití je podle [14] detailní specifikace funkčních požadavků. Jednotlivé funkční požadavky definované v analýze požadavků se většinou rozdělí na více případů užití.

Případy užití se dají dále použít jako základ pro tvorbu uživatelské příručky nebo další zpřesnění odhadů pracnosti. Dále slouží už jako zadání pro programátory tvořící aplikaci.

Model případů užití se skládá z následujících tří hlavních sekcích:

- **Seznam účastníků** je seznam všech uživatelů, kteří mohou k aplikaci přistupovat.
- **Diagramy případů užití** zobrazuje propojení účastníka s případy užití.
- **Seznam případů užití** obsahuje detailní informace ke všem případům užití, scénáře (hlavní a alternativní) a podmínky provedení.

Pro větší přehlednost seznam případů užití vždy zahrnu pod diagram, kde se daný případ nachází. Scénáře budu dělat pouze u složitějších případů užití.

2.4.1 Seznam účastníků

Aplikace bude obsahovat pouze 2 účastníky:

- **Anonymní uživatel**

  Nemá přístup do hlavních funkcí aplikace. Může si pouze zobrazit informace o aplikaci, registrovat se nebo se přihlásit.

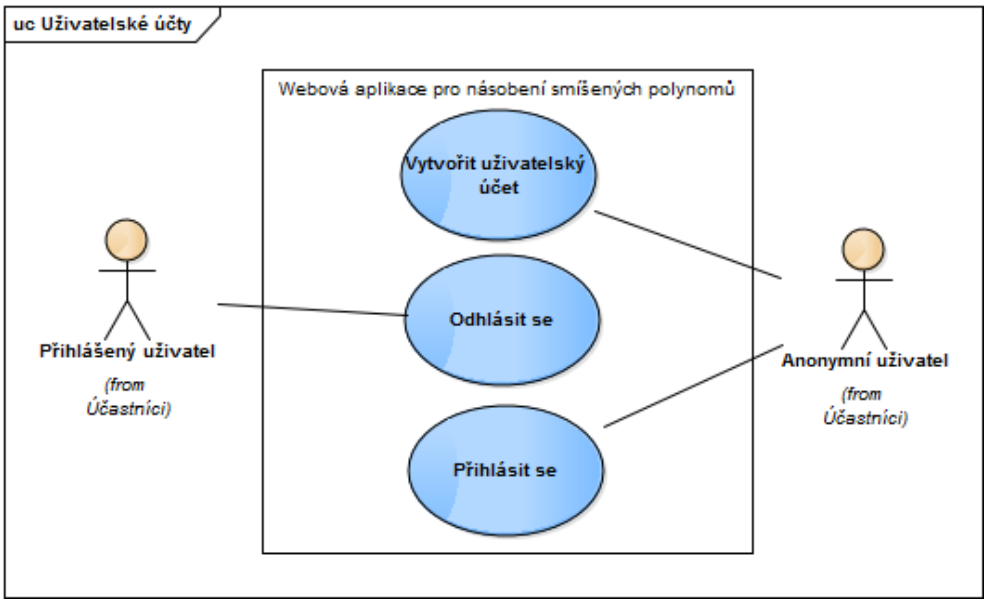
- **Přihlášený uživatel**

  Má přístup ke všem hlavním funkcím aplikace. Může se starat o své polynomy (generování/nahrání/smazání) a výpočty (provedení/smazání). Dále se může odhlásit nebo zobrazit informace o aplikaci stejně jako anonymní uživatel.

2.4.2 Diagramy případů užití

Uživatelské účty

Tento model případů užití je relativně jednoduchý, proto není potřeba vytvářet jednotlivé scénáře případů užití. Přihlášený uživatel má dostupnou funkci odhlášení a uživatel, který přistupuje ke stránce anonymně se může buď přihlásit nebo zaregistrovat. Tento model zobrazuje diagram na obrázku 2.2.



Obrázek 2.2: Diagram případů užití pro uživatelské účty

Operace s polynomy

Další diagram případů užití zobrazený na obrázku 2.3 zachycuje operace s polynomy uživatele. Uživatel má k dispozici generátor polynomů, kde zadá název a velikost polynomu a procentuální počet nul. Dále si může nahrát soubor s vlastním polynomem ve přesně specifikovaném formátu. Přes zobrazení seznamu polynomů jsou dostupné tři další funkce a to upravení polynomu zahrnující smazání a úpravu názvu, stáhnutí polynomu a zobrazení detailu polynomu.

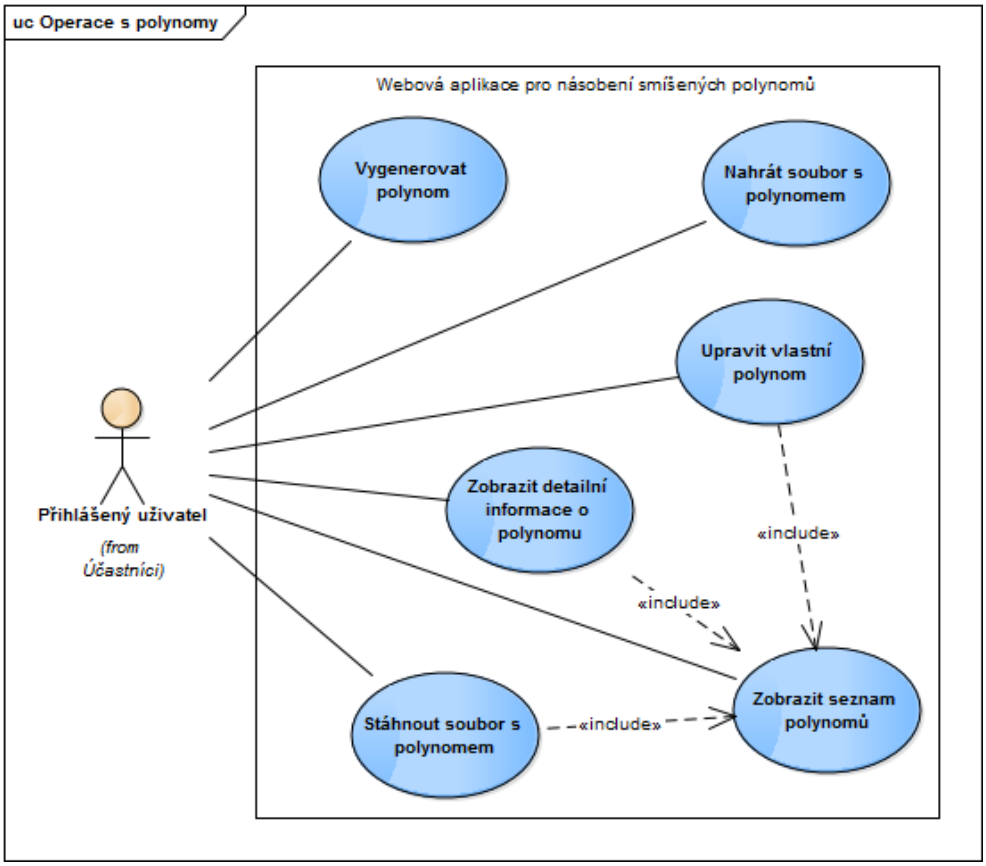
Většina scénářů případů užití zobrazených na diagramu je přímočará v tom smyslu, že si uživatel vybere funkci z menu, vyplní potřebné informace do formuláře a odešle je. Výjimku tvoří případ užití stáhnutí souboru s polynomem, proto popíši i scénáře vztahující se k tomuto případu.

Hlavní scénář: stáhnutí souboru s polynomem:

1. Scénář případu užití začíná, když uživatel potřebuje stáhnout soubor se svým polynomem.
2. Uživatel si zobrazí seznam všech jeho polynomů, který se vygeneruje ve formě tabulky. V posledním sloupci tabulky, kde jsou zobrazeny akce, je odkaz na stažení polynomu.
3. Po kliknutí na odkaz prohlížeč automaticky začne daný polynom stahovat.

Alternativní scénář: stáhnutí souboru s polynomem:

1. Scénář případu užití začíná ve druhém kroku hlavního scénáře, když si uživatel zobrazí seznam všech svých polynomů.
2. Uživatel klikne na zobrazení detailu polynomu a aplikace vygeneruje data v detailu, kde se bude nacházet i odkaz ke stažení daného polynomu.
3. Po kliknutí na odkaz prohlížeč automaticky začne daný polynom stahovat.



Obrázek 2.3: Diagram případů užití pro operace s polynomy

Výpočty

Poslední diagram případů užití na obrázku 2.4 zobrazuje, co vše může uživatel dělat s výpočetní částí. Může provést výpočet nad dvěma svými polynomy a přes seznam proběhnutých výpočtů má dostupné funkce upravení výpočtu a porovnání dvou výpočtů. Upravení je stejně jako u polynomů smazání výpočtu

a porovnání spočívá v porovnání rychlostí celkových výpočtů na různém počtu vláken a v poměrech načítání vstupu, výpočtu a ukládání výstupu.

Větší pozornost věnuji provedení výpočtu a porovnání výpočtů, proto u nich vytvořím i scénáře. U provedení výpočtu je pouze hlavní scénář a při porovnávání výpočtů je hlavní i alternativní. U zobrazení detailu a úpravy provedených výpočtů jsou scénáře jednoduché ve smyslu, že uživatel zobrazí seznam výpočtů, zvolí danou akci a dostává kýžený výsledek.

Hlavní scénář: provedení výpočtu:

1. Scénář případu užití začíná, když se uživatel rozhodne vynásobit 2 polynomy.
2. Aplikace zobrazí 2 seznamy polynomů uživatele. Z každého seznamu bude moci uživatel vybrat jeden polynom a následně přejít k dalšímu kroku.
3. Uživatel vybere jeden algoritmus ze seznamu podporovaných algoritmů, zvolí název polynomu a určí počet vláken pro provedení výpočtu. Při vybírání počtu vláken může být vybráno více možností. Pokud se tak stane, výpočet je proveden několikrát (tolikrát, kolik bylo vybráno možností) na vybraných počtech vláken.
4. V posledním kroku uživatel zkontroluje vyplněné informace a potvrdí je. Je možné se vracet na předchozí kroky kvůli úpravě vybraných informací. Tato možnost je dostupná i u předchozích kroků.

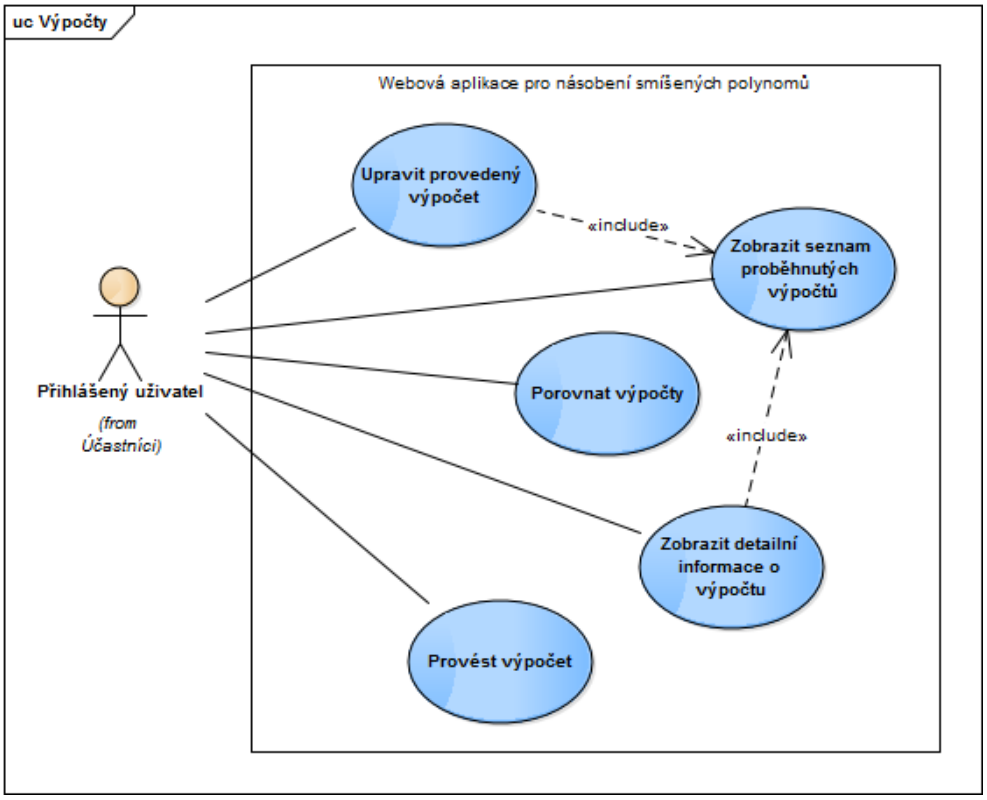
Hlavní scénář: porovnání výpočtů:

1. Scénář případu užití začíná, když se uživatel rozhodne porovnat 2 výpočty.
2. Aplikace zobrazí 2 seznamy provedených výpočtů. Z každého seznamu si uživatel vybere jeden.
3. Zobrazí se stránka s porovnáním výpočtů zvolených v předchozím kroku. Porovnání spočívá v porovnání rychlostí na různém počtu vláken a poměru načítání vstupu, zapsání výstupu a provedení samotného výpočtu.

Alternativní scénář: porovnání výpočtů:

1. Scénář případu užití začíná již v prvním kroku hlavního scénáře, kdy se uživatel rozhodne porovnat 2 výpočty.
2. Uživatel si zobrazí detail provedeného výpočtu. V detailu bude možnost porovnání tohoto výpočtu s jiným.

3. Po kliknutí na tuto možnost se zobrazí seznam provedených výpočtů, ze kterých uživatel vybere jeden a systém vygeneruje porovnání.



Obrázek 2.4: Diagram případů užití pro výpočty

2.5 Kontrola splnění všech požadavků

Tabulka 2.2 zobrazuje pokrytí všech funkčních požadavků případy užití. Pokud by se zde nacházel případ užití, který by nepokrýval žádný funkční požadavek, tak by musel být odstraněn. Na druhou stranu pokud by tu byl funkční požadavek, který by nebyl pokryt žádným případem užití, tak by se musel přidat případ užití pokrývající tento požadavek.

Jak můžeme vidět v posledním řádku tabulky, všechny funkční požadavky jsou pokryty a žádný případ užití zde není navíc, takže je vše v pořádku.

Tabulka 2.2: Tabulka pokrytí funkčních požadavků případy užití

	F1	F2	F3	F4	F5	F6	F7	F8	F9
Nahrát polynom		X							
Stáhnout polynom					X				
Upravit vlastní polynom			X						
Vygenerovat polynom				X					
Zobrazit detail polynomu			X						
Zobrazit seznam polynomů		X	X		X				
Porovnat výpočty								X	
Provést výpočet						X			
Zobrazit detail výpočtu							X		
Upravit provedený výpočet									X
Zobrazit seznam výpočtů							X	X	
Přihlásit se	X								
Odhlásit se	X								
Vytvořit uživatelský účet	X								
Splněno	✓	✓	✓	✓	✓	✓	✓	✓	✓

2.6 Doménový model

Doménový model zobrazen na obrázku 2.5 vychází z předchozí analýzy požadavků 2.4, kde byly definovány nároky a požadavky na výslednou aplikaci.

Analytický doménový model jako takový nemá zobrazovat atributy potřebné pro celu realizaci aplikace, ale jen atributy sloužící k pochopení celého problému. To je důvod, proč můj model neobsahuje například vlastní a cizí klíče.

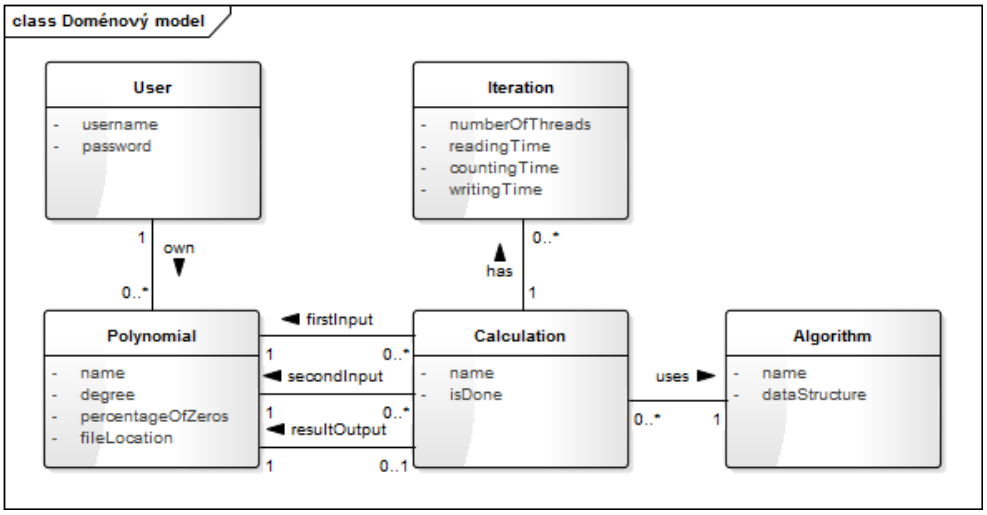
Cíle analytického doménového modelu jsou dle [15] následující:

- Popsat data
- Popsat vazby mezi entitami
- Identifikovat stavy entit
- Vytvořit základ pro design (databázový model, model tříd)
- Popsat význam termínů
- Zachytit atributy

Většinu z těchto cílů zachycuje obrázek 2.5 a pod ním popis jednotlivých entit. Výjimku tvoří identifikace stavů entit, která se typicky znázorňuje dalším diagramem, tzv. stavovým diagramem. V mé aplikaci je pouze jedna entita, která má nějaký stav a to je entita Calculation. Stavový diagram této entity je znázorněn na obrázku 2.6.

2.6.1 Entity

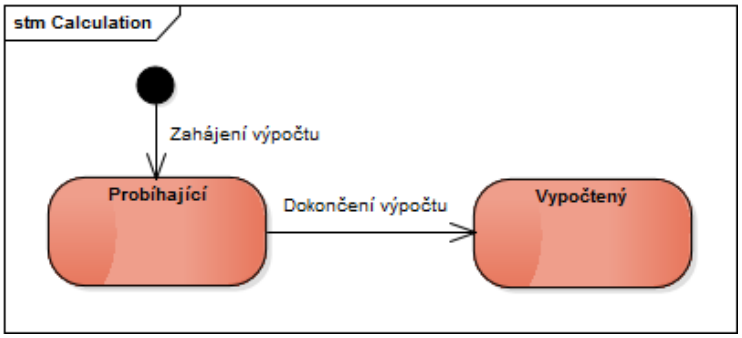
Entita User představuje uživatele aplikace. Uživatel vlastní nějaké polynomy, které zastupuje entita Polynomial. Samotný výpočet zobrazen entitou Calculation má na vstupu 2 polynomy (vztahy first input a second input) a jeho výsledkem je jeden polynom (vztah resultOutput). Každý výpočet používá jeden algoritmus, který zastupuje entita Algorithm, a může proběhnout několikrát s různým počtem vláken, což představuje entita Iteration. Za povšimnutí stojí fakt, že může proběhnout výpočet a nemít ani jednu iteraci. Tím se zobrazuje probíhající výpočet a jednotlivé iterace mu budou přidány až po jeho doběhnutí.



Obrázek 2.5: Doménový model

2.6.2 Stavové diagramy

V mé aplikaci je jen jedna entita se stavy a to je entita Calculation představující výpočet. Jelikož výpočet může trvat i desítky sekund, tak probíhá na serveru a až po jeho dokončení jsou do databáze doplněny všechny informace.



Obrázek 2.6: Stavový diagram pro entitu Calculation

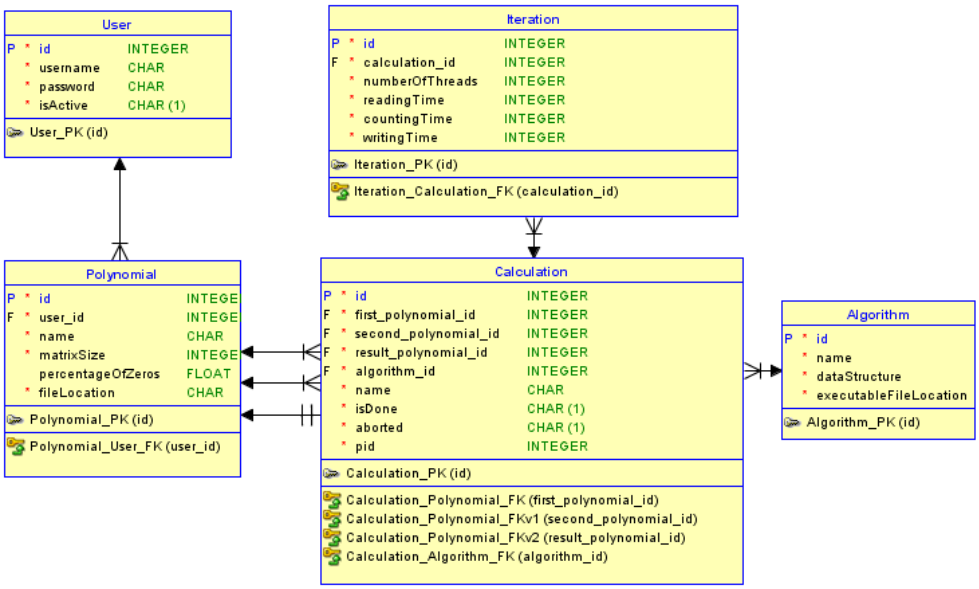
Při zahájení výpočtu je výpočet ve stavu „Probíhající“ a po jeho dokončení se přesune do stavu „Vypočtený“.

Kapitola 3

Návrh

3.1 Konceptuální model databáze

Konceptuální model databáze na obrázku 3.1 vychází z doménového modelu definovaného v kapitole 2.6. Na rozdíl od něj už obsahuje všechny klíče tabulek a typy atributů.



Obrázek 3.1: Konceptuální model databáze

3.2 Návrh grafického uživatelského rozhraní

Wireframe je 2 dimenzionální reprezentace výsledné webové stránky [17].

Slouží hlavně pro ujasnění umístění jednotlivých prvků v prostoru stránky. Z toho důvodu většinou neobsahuje žádné barvy ani obrázky. Místo obrázků se na stránku umísťuje proškrtnutý obdélník. Pro návrh grafického uživatelského rozhraní (GUI) jsem použil právě wireframe, které jsem vytvořil v programu Pencil [5].

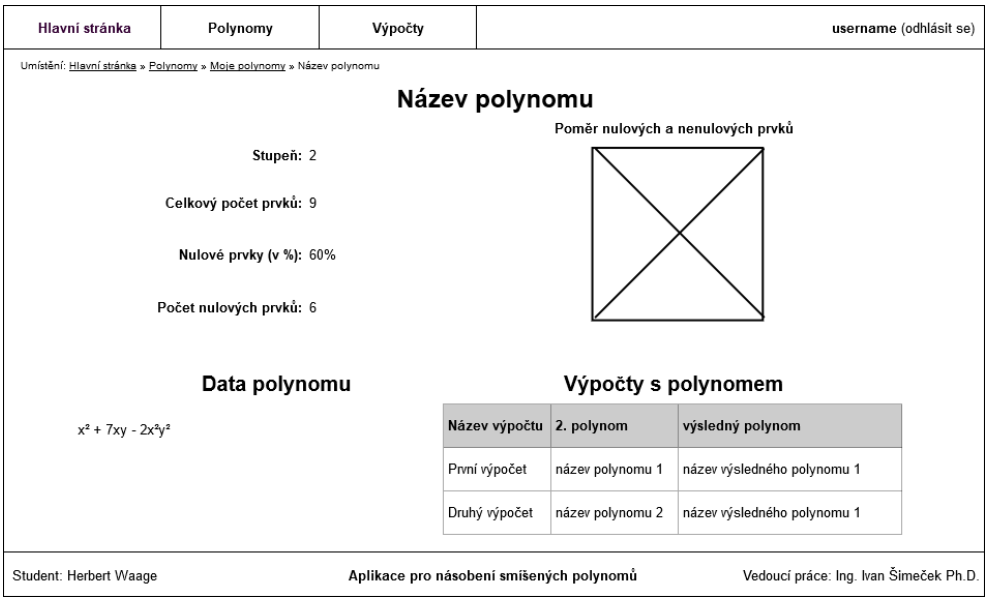
Vytvořil jsem wireframe jen pro stěžejní části výsledné aplikace, které popíši ve zbytku kapitoly. Při jejich vytváření jsem dbal na funkční požadavky na aplikaci, které byly definovány v analýze.

3.2.1 Detail polynomu

Jelikož se jedná o první wireframe, tak zde popíši i celkovou strukturu, kterou obahuje každá stránka.

Na obrázku 3.2 můžeme vidět, že aplikace obsahuje menu umístěné nahoře vlevo. První položka menu nebude mít žádné podmenu, ale další dvě položky budou mít svá rozbalovací podmenu. Vpravo nahoře lze vidět přihlašovací jméno přihlášeného uživatele a také odkaz na odhlášení. Pod menu je umístěna drobečková navigace umožňující sledovat uživateli, kde se právě nachází. Aplikace také obsahuje patičku umístěnou dole.

Samotný detail obsahuje obecné informace o polynomu umístěné vlevo. Napravo od nich je graf zobrazující poměr nulových a nenulových prvků. Pod grafem je tabulka výpočtů, kde byl daný polynom použit. Vlevo od tabulky jsou pak data polynomu.

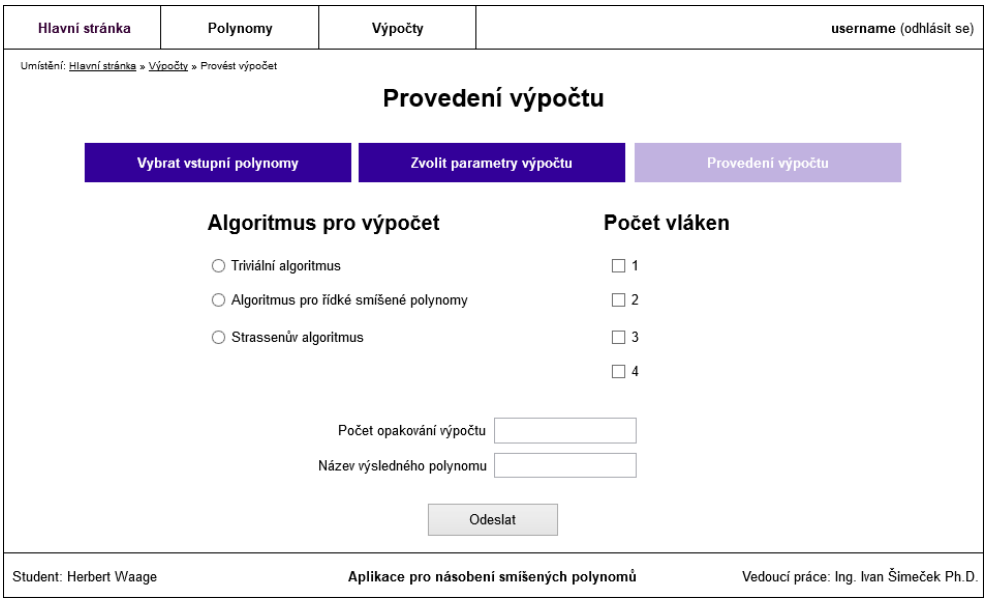


Obrázek 3.2: Wireframe pro detail polynomu

3.2.2 Výpočet

Výpočet je rozdělen do třech částí. V první části se vybírají vstupní polynomy, v druhé pak parametry výpočtu a v poslední se zkontrolují zadané informace a spustí se výpočet. Z důvodu úspory místa a přehlednosti práce zde popíši pouze druhou část. Ostatní části jsou na přiloženém médiu.

Obrázek 3.3 zobrazuje wireframe pro druhý krok provedení výpočtu. Nahoře se zobrazuje průběh celé akce odlišený barevně, proto tento wireframe obsahuje i barvy, což většinou wireframy neobsahují. V levé části si uživatel vybírá ze seznamu podporovaných algoritmů a v pravé části určuje na kolika vláknech výpočet poběží. Pod těmito informacemi jsou na středu umístěny dva textboxy určující počet opakování výpočtu pro věrohodnější výsledek a název výsledného polynomu. Nakonec je zde tlačítko pro odeslání celého formuláře.

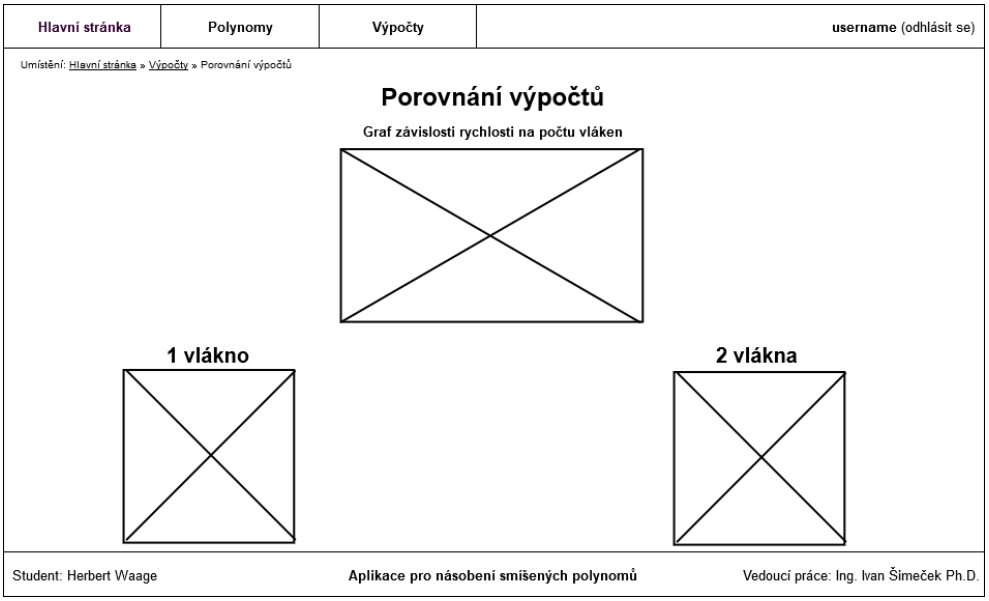


Obrázek 3.3: Wireframe pro druhý krok provedení výpočtu

3.2.3 Porovnání výpočtů

Pokud uživatel bude chtít porovnat dva výpočty, pak se mu zobrazí dva seznamy jeho provedených výpočtů. Po vybrání jednoho výpočtu z každého seznamu se mu vygeneruje porovnání, které představuje wireframe 3.4.

Porovnání obsahuje graf závislosti počtu vláken na rychlosti algoritmů, ve kterém jsou oba výpočty vykresleny zároveň. Následně je seznam počtu vláken a pro každé vláknu je vykreslen graf poměrů načítání dat, výpočtu a uložení dat.



Obrázek 3.4: Wireframe pro porovnání výpočtů

3.3 Použité implementační nástroje

V následujícím textu popíši technologie a nástroje, které jsem použil při implementaci. Co se týče webového rozhraní, tak se jedná o PHP framework Symfony [20] známý pro jeho robustnost a javascriptovou knihovnu Google Charts [7] sloužící pro vykreslování grafů. Při implementaci algoritmů v jazyku C++ jsem použil technologii OpenMP zajišťující rozložení výpočtů do více vláken a tím zrychlení celého procesu výpočtu. Všechny tyto technologie jsem měl dané zadáním, takže jsem si nemohl vybírat.

3.3.1 OpenMP

Pro paralelizaci algoritmů existují 2 hlavní technologie: OpenMP [6] a Posix [11] standard pro použití vláken. Použil jsem technologii OpenMP, kde je jednodušší použití na rozdíl od Posixu.

V Posixu se dají řídit jednotlivá vlákna na nízké úrovni abstrakce. Dá se vytvořit vlákno a řídit celý jeho životní cyklus, uzamykání kritických sekcí pomocí semaforů či zámků. To se hodí pro aplikace, ke kterým přistupuje více uživatelů, jako jsou například serverové aplikace.

V OpenMP se využívá tzv. direktiv, pomocí kterých se určuje, co poběží paralelně a co bude sdílená proměnná. Tato technika se používá pro programování aplikací, ve kterých se vykonává stejný kód nad velkým množstvím dat, což je přesně případ násobení smíšených polynomů.

Možnosti OpenMP

Při tvorbě této sekce jsem čerpal z [2]. OpenMP má velké množství direktiv, pomocí kterých se dají řídit vlákna. Já zde popíši jen pár hlavních. Označení direktivy v C/C++ programu probíhá následujícím způsobem:

```
#pragma omp directive-name [clause[[,] clause]...]
```

Direktiva vždy začíná **#pragma omp**, po kterém následuje název direktivy s možnou jednou nebo více možnostmi.

- **parallel for**: Při paralelizaci programu chceme většinou zrychlit část programu, která probíhá nejdéle, aby došlo k co největšímu zrychlení. Nejdéle probíhají nejčastěji cykly a právě pro tento účel slouží direktiva `parallel for`. Ve výchozím stavu tato direktiva paralelizuje jeden for cyklus, který je nejvíce vně. Pokud chceme paralelizovat více vnořených cyklů, tak k tomu slouží možnost `collapse(n)`, kde n je počet cyklů, které se mají paralelizovat. Další možnost, kterou jsem využil i já při analýze výpočtů, je `num_threads(n)`. To určuje přesné číslo n, které udává, na kolika vláknech má být výpočet spuštěn.

- **shared, private**: Pokud paralelizujeme nějaký cyklus, tak určujeme, co budou mít všechny smyčky jako sdílenou proměnou (`shared`) a co bude mít každý jako vlastní proměnou (`private`). Ve výchozím nastavení je sdílená proměná taková, která je definována mimo cyklus. Vlastní proměnné jsou pak proměnné definované uvnitř cyklu. Toto nastavení se dá změnit právě pomocí možností `shared(list)` a `private(list)`.

- **critical**: Někdy je potřeba definovat sekce, do kterých bude moci přistupovat jen jedno vlákno. Takové sekce se nazývají *kritické sekce* a pro jejich označení slouží direktiva `critical`. Typicky jsou to sekce, kde se ukládají složitě vypočtené výsledky do sdílené proměnné.

- **atomic**: Pokud nechceme definovat celou kritickou sekci, tak můžeme říci, aby proběhl následující příkaz *atomicky*. Atomicky znamená, že při zahájení příkazu nesmí zahájit příkaz jiné vlákno. K tomu slouží právě direktiva `atomic`.

3.3.2 Symfony

Symfony je open source PHP framework, který zjednodušuje proces celého vývoje webové aplikace. Díky rozdělení aplikace do tří hlavních částí (řídící logika, datový model a prezentace dat uživateli) podporuje budoucí rozšiřitelnost. Řídící logiku zajišťuje PHP, datový model objektově relační mapování pomocí Doctrine a prezentaci dat uživateli šablonovací systém Twig.

Symfony je podporováno firmou SensioLabs, což je firma se 13ti lety zkušeností v oblasti vývoje webových aplikací. Navíc Symfony používá velká část

programátorské komunity, což ukazuje i fakt, že za měsíc je framework stáhnut přibližně miliónkrát.

3.3.2.1 PHP

Řídící logiku zajišťuje v Symfony PHP (Hypertext Preprocessor), což je široce používaný scriptovací jazyk běžící na serveru. Hlavní účel PHP je především programování dynamických webových stránek a aplikací například ve formátu HTML nebo XML.

Základní podpora objektového programování byla přidána již ve verzi PHP 3. Ale až verze PHP 5 rozšířila tuto funkcionalitu natolik, že se objevila řada frameworků využívajících potenciál objektového programování jako například již zmíněný framework Symfony, velmi rozšířený Nette nebo Zend Framework.

3.3.2.2 Doctrine

Komunikaci s databází zajišťuje v Symfony objektově relační mapování pomocí Doctrine [22]. Jedná se o soubor PHP knihoven, které zajišťují persistenci dat.

Doctrine je založeno na anotacích, pomocí nichž určujeme, zda má být entita a její atributy uloženy do databáze a jakého mají být typu. Příklad takového mapování můžeme vidět na následující ukázce:

```
//...
use Doctrine\ORM\Mapping as ORM;
/**
 * ...
 * @ORM\Table(name="Polynomial")
 * ...
 */
class Polynomial {
    //...
    /**
     * @ORM\Column(type="string", length=64)
     */
    private $name;
    //...
}
```

Můžeme vidět, že se dá pomocí jednoduché anotace určit název tabulky a typ sloupce výsledné tabulky. Po této definici stačí Doctrine říct, že má vytvořit schéma v databázi a následně už s entitou mohu pracovat a nemusím se skoro starat o její persistenci. Skoro protože musím Doctrine říci na konci úprav dané entity, aby se uložila, což se v Symfony provádí pomocí entity manageru.

3.3.2.3 Twig

Prezentaci dat uživateli zajišťuje v Symfony šablonovací systém Twig [13], což je velice rozšířený šablonovací systém díky následujícím vlastnostem:

- **Rychlost**: Twig zkompiluje šablonu vytvořenou programátorem do optimalizovaného PHP kódu, kde je režie snížená na minimum.
- **Bezpečnost**: Twig obsahuje sandbox mód pro vyhodnocování nedůvěryhodného kódu. To jsou například šablony, kde mohou obsah ovlivňovat uživatelé a na základě toho i útočit na danou stránku.
- **Flexibilita**: Twig umožňuje programátorům definovat vlastní tagy a filtry pomocí rozšíření.

Twig také podporuje dědičnost šablon, což usnadňuje znovupoužití kódu například pro menu v aplikaci.

3.3.2.4 Autentizace a autorizace

Jak autentizaci, tak autorizaci (princip popsaný v kapitole 2.1.1) lze v Symfony provádět několika možnými postupy. U autentizace je to například přihlašovací formulář, který jsem zvolil při implementaci já (v kapitole 4.1.2) nebo autentizace pomocí X.509 certifikátu. Autentizaci lze řešit například pomocí *access control* listů definovaných v konfiguračním souboru security.yml nebo zabezpečením objektů a metod.

3.3.3 Google Charts

Google charts je javascriptová knihovna od společnosti Google sloužící pro vykreslování grafů. Podporuje řadu typů grafů od základních sloupcových a spojnicových grafů až po graf ve formu časové osy nebo hierarchické grafy například struktury organizace.

Použití knihovny je vcelku jednoduché, stačí zvolit typ grafu, předat data a určit kam se má daný graf vykreslit. K vykreslování grafů se používá technoligie HTML5/SVG (pro starší verze prohlížeče Internet Exploreru je zabudován VML) pro zajištění multiplatformní podpory a podpory všemi prohlížeči.

Kapitola 4

Realizace a testování

4.1 Implementace

V následujících podkapitolách popíši, jakým způsobem jsem implementoval výslednou aplikace.

4.1.1 Menu

Jelikož menu je jeden ze základních prvků aplikace, tak jsem ho zahrnul ho hlavní Twig šablony, od které dědí všechny ostatní šablony. To zajišťuje přítomnost menu na každé stránce aplikace.

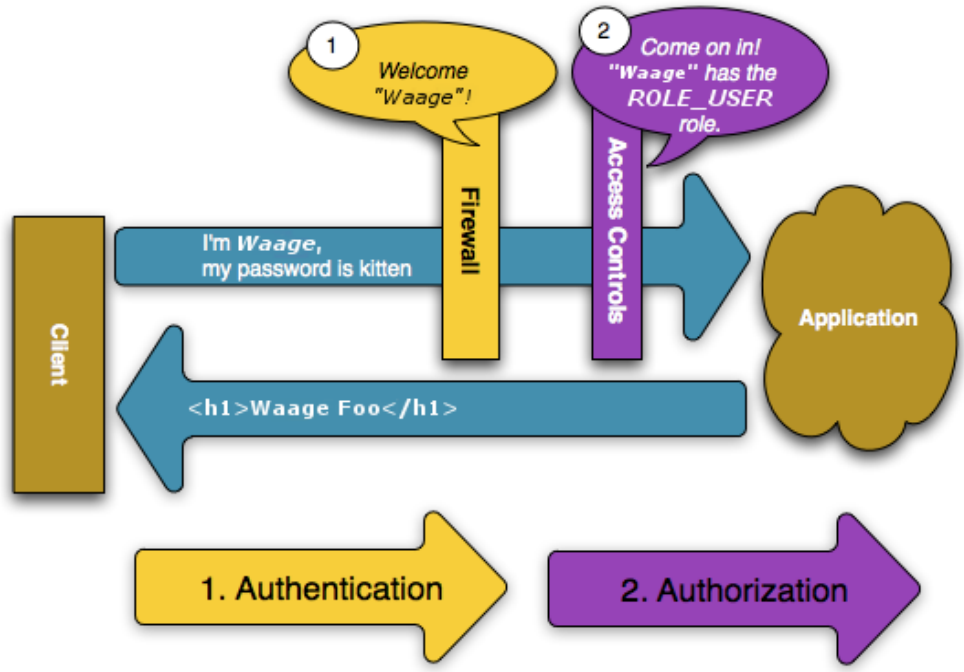
Dbal jsem i na absenci výskytu cyklických odkazů v menu, které mohou mást uživatele. Pokud se totiž uživatel vyskytuje na jedné stránce a klikne na odkaz, tak očekává, že přejde na jinou stránku. Když přejde na stejnou stránku, tak může dojít k jeho dezorientaci. Tuto funkcionalitu zajišťuje mnou vytvořené rozšíření Twigu s názvem `MenuExtension`. Jedná se o funkci, která na základě názvu, cesty k cíli a aktivní položky menu předané z kontrolleru do šablony vytvoří položku menu buď s odkazem, nebo bez odkazu.

Menu je víceúrovňové a objevení podmenu je zajišťováno javascriptem. To může vyvolat potíže, protože javascript běží na straně klienta (prohlížeče). Pokud má tedy uživatel v prohlížeči javascript vypnutý nevyjede mu podmenu. Tento problém jsem vyřešil mezistránkou, která obsahuje odkazy na všechny položky podmenu. Odkaz na tuto mezistránku je umístěn v položce hlavního menu, která je viditelná pořád.

4.1.2 Autentizace a autorizace

Pro zajištění autentizace jsem zvolil metodu přihlašovacího formuláře, který je jedním z nejběžnějších způsobů autentizace na webových stránkách. Jelikož v mé aplikaci má každý uživatel přístup ke všem možnostem aplikace (není implementována administrátorská sekce), tak jsem autorizaci nemusel řešit.

V celé mé aplikaci kromě hlavní stránky a stránek s registrací a přihlašovacím formulářem probíhá celý proces podle schématu zobrazeném na obrázku 4.1. Po přihlášení se uživatel „Waage“ s heslem „kitten“ dostane při načtení každé stránky k firewallu, který přihlášený uživatel vždy projde. Dále se dostane ke kontrole přístupu, kterou v mé aplikaci projde každý přihlášený uživatel, protože nemám definovanou jinou roli než `ROLE_USER`. Poté aplikace vytvoří příslušnou odpověď a pošle ji zpět.



Obrázek 4.1: Autentizace a autorizace

4.1.3 Zpracování formulářů

Formuláře jsou používány ke zpracování uživatelského vstupu. Symfony má relativně jednoduchý způsob vytváření a zpracování formulářů včetně jejich validace.

Pro vytvoření formuláře jsem vytvořil nejdříve třídu, do které jsem přidal atributy reprezentující uživatelský vstup. Dále jsem vygeneroval třídu pro daný formulář, ve které jsem určil jak má vypadat uživatelský vstup (jestli to má být textové pole, tlačítko nebo něco jiného). Poté stačí v controlleru vytvořit daný formulář a poslat do šablony, kde se vykresluje.

Zpracování formuláře probíhá přes metodu daného formuláře `handleRequest`, která zajistí namapování uživatelského vstupu na objekt. Dále se testuje zda je formulář potvrzený a validní. Validace formuláře se v Symfony zajišťuje po-

mocí anotací nad atributem třídy, kam má být uživatelský vstup namapován.

4.1.4 Integrace C/C++ programů

Protože jednotlivé algoritmy jsou psané v C/C++, tak jsem s nimi musel nějak pracovat z PHP prostředí. Symfony nabízí komponentu `Process` [21], která zajišťuje spuštění příkazu v podprocesu. Tato komponenta umožňuje mimo jiné sledovat průběh procesu, zjišťovat stav procesu a získat výstup z procesu. Pro tyto její vlastnosti jsem ji použil při spuštění generátoru polynomů a validátoru polynomů, kteří běží krátkou dobu. Vždy si počkám na jejich dokončení a poté přesměruji uživatele na následující stránku.

Pro spouštění samotných algoritmů používám PHP funkci `shell_exec` [8], které předám textový řetězec s cestou programu, parametry programu, indikaci pro běh na pozadí (&) a přesměrování standartního a chybového výstupu do /dev/null. To mi zajistí běh programu na pozadí i po načtení další stránky uživatelem. Do databáze si uložím `PID` (Process Identifier) a při zobrazování seznamu výpočtů detekuji, jestli proces ještě běží, což zjistím právě pomocí PID. Pokud již neběží, tak se kouknu na soubor s průběhem výpočtu, kam proces ukládá jak dlouho na kolika vláknech co běželo a pokud je výpočet hotový, tak zpracuji informace. Pokud proces neběží a v souboru s průběhem výpočtu není uvedeno, že byl dokončen, tak informuji uživatele o tom, že jeho výpočet byl násilně ukončen. To se stává většinou pro dlouhou dobu běhu procesu.

4.1.5 Použité algoritmy

Teoretická část implementovaných algoritmů je popsána v kapitole 1.2. V následujících sekcích popíši už přesnou formu implementace v jazyce C/C++ a způsob jak jsem paralelizoval jednotlivé algoritmy. Pro paralelizaci jsem použil technologii OpenMP popsanou v kapitole 3.3.1.

4.1.5.1 Triviální algoritmus

Základní implementace První algoritmus, který jsem implementoval, je triviální algoritmus. Jeho výsledky mi následně sloužili pro otestování správnosti dalších implementovaných algoritmů.

Polynomy načtu ze souborů a jednotlivé hodnoty uložím do 2D pole, kde první index reprezentuje řádek matice polynomu a druhý index představuje sloupec matice polynomu. Ukládám tam tedy i nulové prvky. Před začátkem celého výpočtu si vytvořím 2D pole pro výsledný polynom, jehož velikost je součet velikostí matic vstupních polynomů zmenšená o 1. Toto pole vyplním nulami, protože budu následně přičítat vypočtené hodnoty k původním hodnotám v matici.

Samotný proces násobení probíhá ve čtyřech vnořených `for` cyklech, kdy první iteruje přes sloupce prvního polynomu, druhý přes řádky prvního po-

lynomu, třetí přes sloupce druhého polynomu a čtvrtý přes řádky druhého polynomu. To mi zajistí vynásobení každého prvku prvního polynomu s každým prvkem druhého polynomu. Výpočet každého násobení přičtu na místo určené součtem indexů řádků a součtem indexů sloupců obou polynomů.

Paralelizace Zde jsem využil metodu paralelizace `for` cyklu `parallel for`. Tuto direktivu jsem umístil nad první `for` cyklus a přidal k ní možnost určující počet vláken pro výpočet `num_threads(n)`, kde si n předávám do funkce realizující výpočet. To mi slouží pro analýzu algoritmů a vykreslování grafů rychlostí jednotlivých výpočtů na různém počtu vláken ve výsledné aplikaci. Paralelizaci vnitřních `for` cyklů jsem nedělal, protože se to nevyplatí [23]. Jediná možnost, kdy by se to vyplatilo, by byla taková, že by první polynom obsahoval méně sloupců než je dostupný počet jader. Což je samozřejmě možné, ale nestojí to za režii paralelizace vnitřních for cyklů u velkých matic polynomů.

Problém zde nastává jak synchronizovat ukládání do proměnné reprezentující výsledný polynom, protože to je jediný složitější výpočet, který zde probíhá. Ostatní jsou inkrementace řádků a sloupců. Vyřešil jsem to tak, že mám `private` proměnnou tmp, do které si uložím výsledek vynásobení. Následně atomicky pomocí direktivy `atomic` přičtu do matice výsledného polynomu hodnotu v proměnné tmp. Jiná možnost by byla rozdělit výsledný polynom na několik částí a zparalelizovat výpočty těchto částí.

4.1.5.2 Algoritmus pro násobení řídkých smíšených polynomů s využitím CRS formátu

Základní implementace Dalším algoritmem, který jsem implementoval je algoritmus, který využívá formátu CRS popsaného v kapitole 1.1.3. Je výhodný hlavně pro řídké smíšené polynomy.

Jelikož nevím, jak dlouhé budou nakonec pole reprezentující formát CRS, tak jsem zde použil C++ třídu `vector`. Pokud `vector` nemá dostatek místa pro uložení dalšího prvku, tak si další místo naalokuje sám. Takže jsem nemusel řešit realokaci pole při nedostatku místa. Podporuje také průchod všech prvků pomocí `iterátoru`, což jsem využil při ukládání výsledků do souboru reprezentující výsledný polynom.

U formátu CRS se špatně detekuje offset řádku, který je nulový. Tento problém jsem vyřešil tak, že první offset nezačíná na 0, jak je definováno ve formátu CRS, ale na 1. K indikaci celého nulového řádku slouží právě nulová hodnota.

Pro výsledný polynom jsem nezvolil formát CRS, ale klasický formát 2D pole i s nulovými prvky. Tento postup jsem zvolil z toho důvodu, že vynásobení dvou smíšených polynomů dává většinou polynom nesmíšený. Další výhoda je, že se do formátu CRS špatně přidává nový prvek, kdyžto při reprezentaci 2D polem se snadno indikuje pomocí 2 indexů.

Vlastní výpočet pak probíhá ve čtyřech vnořených `for` cyklech, kde první iteruje přes offsety řádků prvního polynomu, druhý přes prvky daného řádku, třetí přes offsety řádků druhého polynomu a čtvrtý opět přes prvky daného řádku ovšem teď druhého polynomu. To mi zajistí vynásobení všech prvků prvního polynomu se všemi prvky druhého polynomu. Ovšem od triviálního algoritmu se tento liší v tom, že nemusíme násobit nulové prvky. Uložení výsledku násobení probíhá stejně jako u triviálního algoritmu přičtením k původní hodnotě výsledného polynomu.

Paralelizace Pro paralelizaci tohoto algoritmu jsem zvolil stejný postup jako při paralelizaci triviálního algoritmu. Zde je ale výhoda, že probíhá mnoho iterování v cyklech mimo atomické uložení výsledku.

Zparalelizoval jsem tedy vnější `for` cyklus, který iteruje přes offsety řádků prvního polynomu. Nastával zde problém, že indexy sloupců byla jedna sdílená proměnná, která se před začátkem nastavila na 0 a uvnitř iterovala. To jsem vyřešil přesunutím proměnné dovnitř cyklu, načež se stala proměnná `private` a na začátku každého cyklu jsem ji nastavil na hodnotu offsetu sníženou o jedničku. Offset musel být snížen o jedničku z důvodu indikace nulových řádků a započetím indexování offsetů od jedničky.

4.1.5.3 Strassenův algoritmus

Základní implementace Strassenův algoritmus slouží pro násobení matic ve specifickém formátu popsaném v kapitole 1.2.3.

Matice načtu ze souborů a uložím do 2D pole. Pokud nevyhovují požadovanému formátu, tak je patřičně rozšířím a nové prvky nastavím na 0. Na začátku si připravím 2D pole i pro výslednou matici, která bude stejně velká jako rozšířené vstupní matice (které jsou stejně velké po rozšíření).

Následně si alokuji místo pro 8 podmatic $A_{i,j}$ a $B_{i,j}$, kde $i, j \in [1, 2]$ a vyplním je patřičnými hodnotami ze vstupních matic **A** a **B**. Poté si naalokuji místo pro 3 pomocné matice k uložení mezivýsledků a 7 matic M_i, které slouží pro vyjádření výsledné matice. Vypočítám všechny matice M_i přesně podle teorie popsané v kapitole 1.2.3. Pro násobení, které je potřeba 7x, využiji Strassenův algoritmus rekurzivně.

Nyní vypočítám 4 podmatice výsledné matice a vytvořím z nich výslednou matici **C**. Nakonec uvolním všechny naalokované prostředky.

Paralelizace Tento algoritmus jsem neparalelizoval, protože ho jen využívá jiný algoritmus popsaný v kapitole 4.1.5.4 počítající násobení smíšených polynomů, který již paralelizovaný je.

4.1.5.4 Algoritmus pro násobení smíšených polynomů s využitím maticového násobení

Základní implementace Tento algoritmus na vstupu očekává matice polynomů stejné velikosti, což není příznivé pro polynomy, které se velikostí matic výrazně liší.

Na začátku načtu polynomy do 2D polí a pokud nejsou stejné velké, tak ten menší patřičně rozšířím na velikost většího. Nově vzniklé prvky nastavím na 0.

Samotný výpočet probíhá v jednom `for` cyklu, který iteruje přes všechny mocniny neznámé x. Pro každou tuto mocninu se předá funkci pro násobení matic ukazatel na příslušný řádek polynomu případně posunutí vpravo při výpočtu mocnin nad půlkou, šířka a výška počítané matice. Prvky vypočtené matice následně uložím na příslušná místa ve výsledném polynomu, jak je popsáno v kapitole 1.3.

Jako algoritmus pro násobení zde používám triviální algoritmus a Strassenův algoritmus popsaný v kapitole 1.2.3.

Paralelizace Pro paralelizaci tohoto algoritmu jsem zvolil paralelizaci `for` cyklu, který iteruje přes všechny mocniny neznámé x. Díky tomu jsem nemusel řešit žádnou synchronizaci sdílené proměnné, kterou je matice výsledného polynomu. A to z toho důvodu, že každý for cyklus počítá pouze jednu mocninu neznámé x, jejíž koeficienty jsou uloženy pouze v jednom sloupci výsledné matice.

4.2 Testování korektnosti algoritmů

Po implementaci všech algoritmů bylo nutné otestovat jejich správnost. K tomu mi sloužil triviální algoritmus, který jsem implementoval jako první.

Napsal jsem 4 programy v C/C++, které zajišťují otestování na náhodných datech a které jsou nahrány na CD přiloženému k práci.

Celé testování je rozděleno do následujících 3 hlavních částí (každou část realizuje jeden program):

Vygenerování polynomů Na začátku se vygeneruje pomocí náhodného generátoru 25 polynomů. Tyto polynomy se generují náhodně o velikosti matice od 1 do 100 a obsahu nulových prvků 0-100%.

Provedení výpočtů V další části se iteruje přes všechny možné kombinace vygenerovaných polynomů, kterých je 625, a pro každou kombinaci se spustí implementované algoritmy. Algoritmy se spouští s použitím 4 vláken, aby byla otestována i správná paralelizace. Každý výsledek výpočtu, kterých je 1875, se uloží do samostatného souboru pro následné otestování správnosti.

Porovnání výsledků V poslední části se iteruje opět přes všechny možné kombinace vygenerovaných polynomů a pro každou kombinaci se porovnají všechny výsledky implementovaných algoritmů mezi sebou.

Poslední program spojuje všechny 3 předchozí, aby byl zajištěn hladký průběh celého testování.

Tento test jsem pustil 3x s úspěšností 100%. Jednou jsem nechal test počítat ne pro 25, ale pro 100 polynomů, tudíž proběhlo 30 000 výpočtů. Výsledky tohoto testu by se ale nevešly na přiložené CD, proto je na něm jen test pro 25 polynomů.

4.3 Uživatelské testování

Při přípravě uživatelského testování jsem vycházel z [10] a [19], kde se popisují všechny fáze provádění testů. Možností testování je veliké množství od testování samotným programátorem přes testování experty až po testování koncovými uživateli. Já jsem zvolil testování výsledné aplikace uživateli, kterému se říká *akceptační testování*.

Vytvořil jsem jednotlivé scénáře pokrývající funkční požadavky na aplikaci a ty jsem předal čtyřem testerům.

Po projití všech scénářů a ověření funkčnosti/nefunkčnosti aplikace byly testerům položeny otázky, které jsou i s výsledky popsány dále v této kapitole.

Scénáře:

1. **Registrujte se a přihlaste se do aplikace.**

   Tento scénář pokrývá funkční požadavek F1 (2.2.1) a vypadá následovně:

   1. Jděte na stránku s testovací verzí aplikace `http://webdev.fit.cvut.cz/~waageher/Bakalarka/`.
   2. Vpravo nahoře klikněte na možnost „Registrace“.
   3. Vyplňte uživatelské jméno, zvolte heslo a potvrďte formulář.
   4. Po přesměrování na stránku s přihlášení, použijte zvolené přihlašovací údaje pro přihlášení do aplikace.

2. **Vygenerujte si 2 polynomy jeden o velikosti matice 150 s obsahem 80% nul a druhý o velikosti matice 100 s obsahem 90% nul.**

   Tento scénář pokrývá funkční požadavek F4 (2.2.4) a vypadá následovně:

   1. Zvolte možnost vygenerovat polynom v generátoru.
   2. Pokud nerozumíte nějakému pojmu, přečtěte si vysvětlivky umístěné vpravo od formuláře pro generování.

3. Zvolte si název polynomu, nastavte rozměr matice a procentuální počet nul.
4. Potvrďte formulář.
5. Celý proces proveďte ještě jednou pro vygenerování druhého polynomu.

3. **Stáhněte soubor s jedním ze svých polynomů.**

   Tento scénář pokrývá funkční požadavek F5 (2.2.5) a vypadá následovně:

   1. Přejděte na seznam svých polynomů přes menu „Polynomy“.
   2. Klikněte na možnost stáhnout polynom v tabulce vpravo od daného polynomu.
   3. Prohlédněte si strukturu souboru, budete to potřebovat u dalšího kroku.

4. **Nahrajte svůj vlastní polynom.**

   Tento scénář pokrývá funkční požadavek F2 (2.2.2) a vypadá následovně:

   1. Zvolte možnost „Nahrát vlastní polynom“ v menu „Polynomy“.
   2. Přečtěte si specifikaci polynomu a vytvořte u sebe na disku soubor s polynomem. Velikost si zvolte jakou chcete (ale velikost souboru je omezena).
   3. Zvolte si název polynomu a vyberte Vámi vytvořený soubor.
   4. Potvrďte formulář.

5. **Váš nahraný polynom je pravděpodobně malý pro demonstraci zrychlení algoritmů, proto ho smažte.**

   Tento scénář pokrývá funkční požadavek F3 (2.2.3) a vypadá následovně:

   1. Přejděte na seznam svých polynomů přes menu „Polynomy“.
   2. Klikněte na křížek v tabulce vpravo od daného polynomu.

6. **Proveďte výpočet se svými polynomy.**

   Tento scénář pokrývá funkční požadavek F6 (2.2.6) a vypadá následovně:

   1. V menu „Výpočty“ zvolte možnost „Provést výpočet“.
   2. Jako první vstupní polynom zvolte jeden ze svých polynomů a jako druhý zvolte ten druhý polynom. Potvrďte formulář.
   3. Jako algoritmus pro výpočet zvolte možnost „Algoritmus s využitím CRS formátu“.
   4. Počet vláken a počet opakování výpočtu nechte ve výchozím nastavení.

5. Zvolte si název výpočtu a název výsledného polynomu. Potvrďte formulář.
6. Zkontrolujte vyplněné údaje. Pokud by nějaký neseděl dříve uvedeným informacím, vraťte se ke špatně provedenému kroku a změňte informace.
7. Zahajte výpočet.
8. Před provedením dalšího kroku vyčkejte na dokončení výpočtu.
9. Celý proces ještě jednou opakujte, ale nyní zvolte jako algoritmus pro výpočet „Triviální algoritmus“. Tento výpočet bude trvat trochu déle, proto mezitím můžete provést další scénář.

7. **Zobrazte si průběh již provedeného výpočtu.**

   Tento scénář pokrývá funkční požadavek F7 (2.2.7) a vypadá následovně:

   1. V menu „Výpočty“ vyberte „Provedené výpočty“.
   2. Přejděte na detail provedeného výpočtu pomocí lupy v tabulce vpravo od daného výpočtu.
   3. Podívejte se na všechna zobrazená data.

8. **Porovnejte dva provedené výpočty.**

   Tento scénář pokrývá funkční požadavek F8 (2.2.8) a vypadá následovně:

   1. Nyní by již měly být oba výpočty dokončené. Pokud tomu tak není, tak vyčkejte na jejich dokončení.
   2. V menu „Výpočty“ vyberte „Porovnání výpočtů“.
   3. Jako první výpočet zvolte jeden z již provedených výpočtů a jako druhý ten zbývající.
   4. Podívejte se na zobrazená data.

9. **Smažte jeden z provedených výpočtů.**

   Tento scénář pokrývá funkční požadavek F9 (2.2.9) a vypadá následovně:

   1. Jděte na seznam všech provedených výpočtů.
   2. Jeden vyberte a ten smažte pomocí křížku v tabulce napravo od daného výpočtu.

4.3.1 Vyhodnocení a dotazník

Po provedení všech scénářů bylo testerům položeno následujících 5 otázek s tím, že první 2 byly povinné a další nepovinné. Pokud bylo potřeba z odpovědí vyvodit nějaké důsledky a upravit aplikaci, tak jsem tyto závěry popsal ihned pod shrnutím odpovědí na danou otázku. Pokud nebylo potřeba vyvodit žádné závěry, tak jsem nic nepopisovat.

- **Jaký webový prohlížeč jste použil?**

  Dva testeři použili webový prohlížeč Google Chrome, jeden Maxthon a jeden Firefox. Tyto prohlížeče až na Maxthon byly definovány v nefunkčních požadavcích jako podporovaní. To tedy zajistilo i otestování funkčnosti a vzhledu aplikace mezi různými prohlížeči.

- **Podařilo se Vám projít všechny scénáře bez sebemenších problémů?**

  Všichni testeři provedli všechny scénáře bez větší problémů. Dva z nich ale měli menší problém u pochopení nahrávání polynomů z vlastních souborů kvůli nejasnému popisu.

  Vyvození důsledků Popis nahrávaných souborů jsem si přečetl a našel jsem tam dokonce i chybu. Celou sekci, kde se formát souboru popisoval, jsem předělal a testerům, kterým s tím měli problém přeposlal, jestli je to již v pořádku. Shodli se na tom, že popis je již dostačující.

- **Co se Vám na aplikaci nejvíce líbilo?**

  Hodně se testerům líbilo grafické zpracování výsledků, výpočtů a porovnání výpočtů, které je zpracováno pomocí Google Charts. Jednomu testerovi se líbilo i intuitivní ovládání aplikace, což byl i jeden z cílů, které jsem si kladl na začátku.

- **Co se Vám naopak vůbec nelíbilo?**

  Dva testeři nechali tuto otázku nevyplněnou. Jednomu se nelíbilo, že po registraci není rovnou přihlášený. Poslednímu testerovi se nelíbilo grafické zpracování potvrzovacích zpráv kvůli jejich nečitelnosti.

  Vyvození důsledků Upravil jsem barvu potvrzovacích hlášek, která byla vůči pozadí v malém kontrastu. Přihlášení ihned po registraci jsem zavrhnul po uvážení, že by to uživatele mohlo dost mást. Na většině stránkách na internetu po registraci uživatel také není přihlášený.

- **Co byste vylepšil nebo dodal do aplikace?**

  Tři testeři by uvítali lepší řešení situace, kdy výpočet ještě probíhá a uživatel má zobrazenou hlášku, že výpočet ještě probíhá. Jeden tester by uvítal někde přečtení smyslu aplikace. Na hlavní stránce je popis, ale podle něj nedostatečný. Další tester by nahradil tlačítko „Odeslat“ za „Další“ u postupování v krocích provedení výpočtu, protože „Odeslat“ ho mate.

Vyvození důsledků Do stránky zobrazující výpočty, kde je také zobrazen stav výpočtům jsem přidal obrázek evokující znovu načtení stránky. Tento obrázek jsem přidal ke každému výpočtu, který momentálně probíhá. Dále jsem upravil popis aplikace na hlavní stránce a nahradil tlačítka „Odeslat“ za „Další“ u prvních dvou kroků provádění výpočtu.

Mimo rozsah otázek jsem si také všiml, že ač je v úkolu napsáno, že mají v generátoru vygenerovat například polynom o velikosti matice 150, tak skoro všichni testeři vygenerovali polynom o velikosti matice 149. To mě přivedlo k úpravě popisu toho, co vlastně znamená velikost matice u generátoru polynomů.

Celkově testování aplikace hodnotím velmi pozitivně. Chyby, které byly nalezeny, se týkaly většinou popisů daných jevů v aplikaci a byly rychle opraveny. Hlavní důvod byl asi ten, že já jako autor aplikace jsem dané jevy již znal. Při jejich popisu neznalým uživatelům jsem proto nekladl velký důraz na jejich vysvětlení a to byla chyba. Nicméně oprava těchto chyb nebyla těžká a nebyl důvod dělat nějaké veliké změny v aplikaci.

4.4 Popis splnění zadání

Následující tabulky popisují splnění určitých bodů zadání. Celkově bylo splněno celé zadání práce.

V tabulce 4.1 je popsáno splnění všech úvodních bodů zadání práce týkajících se obecných věcí.

Tabulka 4.1: Splnění úvodních bodů zadání

Navrhněte webovou aplikaci, která bude porovnávat algoritmy pro násobení smíšených polynomů.	
✓	Tento bod byl splněn a analýza s návrhem jsou popsány v kapitolách 2 a 3.
Implementujte ji ve frameworku Symfony (řídící logika - PHP, datový model - Doctrine, uživatelské rozhraní - Twig).	
✓	Celá aplikace byla implementována ve frameworku Symfony, který je popsán v kapitole 3.3.2.

V tabulce 4.2 je popsáno splnění všech bodů zadání práce týkajících se přímo webové části aplikace.

Tabulka 4.2: Splnění webové části zadání

Navrhněte a implementujte pro ni grafické uživatelské rozhraní, které bude umožňovat:	
✓	Grafické uživatelské rozhraní bylo implementováno a jeho návrh je popsán v kapitole 3.2.
1. přihlášení	
✓	Přihlášení do aplikace je popsáno v kapitole 4.1.2.
2. generování polymonů podle zadaných parametrů (stupeň, počet nenulových hodnot, ...)	
✓	Implementoval jsem program v C++ umožňující generovat polynomy podle zadaných parametrů. V práci jsem ho blíže nepopisoval.
3. uložení a načtení polynomů do/ze souborů ve specifikovaném formátu	
✓	U každého polynomu je možnost si stáhnout jeho soubor. Načtení polynomů probíhá přes validátor a pokud projde, tak se daný polynom úspěšně nahraje.
4. volbu algoritmu pro násobení	
✓	Před samotným zahájením výpočtu si uživatel může zvolit algoritmus pro výpočet.
5. zobrazení výsledku	
✓	Jakmile výpočet proběhne, je vytvořen nový polynom uživatele, který má svůj detail.
6. seznam proběhnutých výpočtů přihlášeného uživatele	
✓	Aplikace umožňuje zobrazit si seznam uživatelových výpočtů.
7. vykreslování grafů rychlostí různých výpočtů (po vybrání ze seznamu proběhnutých výpočtů) s využitím javascriptové knihovny Google Charts	
✓	Aplikace vykresluje grafy rychlostí výpočtů a grafy poměrů načítání dat/výpočtu/uložení dat. Je také možno 2 výpočty mezi sebou porovnat, což se provádí pomocí grafu, kde jsou zobrazeny rychlosti obou výpočtů a pod ním jsou poměry načítání dat/-výpočtu/uložení dat. Vše probíhá pomocí javascriptové knihovny Google Charts.

V tabulce 4.3 je popsáno splnění všech bodů zadání práce týkajících se implementace algoritmů pro násobení.

Tabulka 4.3: Splnění implementace algoritmů

V jazyce C++ implementujte následující algoritmy pro násobení smíšených polynomů:	
✓	Všechny algoritmy pro násobení jsem implementoval v jazyce C++.
a. triviální algoritmus	
✓	Teoretický základ triviálního algoritmu je popsaný v kapitole 1.2.1 a implementace v kapitole 4.1.5.1.
b. Strassenův algoritmus	
✓	Teoretický základ Strassenova algoritmu je popsán v kapitole 1.2.3 a jeho implementace v kapitole 4.1.5.3. Jeho následné využití pro násobení smíšených polynomů je popsáno v kapitole 1.3 a implementace tohoto využití je popsána v kapitole 4.1.5.4
c. algoritmus pro násobení řídkých smíšených polynomů ve formátu CRS (Compressed Row Storage)	
✓	Teoretický základ pro tento algoritmus je popsán v kapitole 1.2.2 a jeho implementace v kapitole 4.1.5.2.

Z předchozích tabulek můžeme vidět, že všechny body zadání byly splněny.

Závěr

Cílem této práce bylo vytvořit webovou aplikaci umožňující porovnávat algoritmy pro násobení smíšených polynomů o dvou neznámých. Byla provedena analýza, návrh a kompletní implementace webové aplikace, která následně úspěšně prošla akceptačními testy.

Pro násobení jsem implementoval triviální algoritmus, algoritmus využívající formát Compressed Row Storage a Strassenův algoritmus. Jednotlivé algoritmy jsou v práci popsány jak po teoretické stránce, tak po stránce implementační. Všechny implementované algoritmy byly také zparalelizovány. Na konci jsem provedl otestování korektnosti implementovaných algoritmů na základě generování náhodných dat.

Hlavním přínosem výsledné webové aplikace je podle mého názoru interaktivita s uživatelem, kdy si může sám vyzkoušet efektivitu algoritmů na různých polynomech lišících se například velikostí nebo celkovým počtem prvků. Navíc si algoritmy může mezi sebou porovnávat.

V budoucnu je několik možností, jak by se dala webová aplikace rozšířit. Jedním z nich je například implementace dalších algoritmů pro násobení nebo přidání administrátorského prostředí.

Literatura

[1] E.J. Barbeau. *Polynomials*. 1st edition. [Online] Naposledy navštíveno 7. 4. 2015. New York: Springer, 2003. ISBN: 0-387-40627-1. URL: http://books.google.com/?id=CynRMm5qTmQC&printsec=frontcover.

[2] Barbara Chapman, Gabriele Jost, and Ruud van der Pas. *Using OpenMP: Portable Shared Memory Parallel Programming*. 1st edition. Cambridge, Massachusetts, USA: The MIT Press, 2007. ISBN: 9780262533027.

[3] World Wide Web Consortium. *Media Queries*. [Online] Naposledy navštíveno 17. 4. 2015. 2015. URL: http://www.w3schools.com/cssref/css3_pr_mediaquery.asp.

[4] Roman Eliáš. "Paralelní násobení řídkých matic". MA thesis. České vysoké učení technické, 2007.

[5] Evolus. *Pencil*. [Online] Naposledy navštíveno 12. 4. 2015. 2012. URL: http://pencil.evolus.vn/.

[6] Richard Friedman. *OpenMP*. [Online] Naposledy navštíveno 22. 4. 2015. 2008. URL: http://openmp.org/wp/about-openmp/.

[7] Google. *Google Charts*. [Online] Naposledy navštíveno 15. 4. 2015. 2015. URL: https://developers.google.com/chart/interactive/docs/.

[8] PHP Group. *PHP - shell_exec()*. [Online] Naposledy navštíveno 28. 4. 2015. 2015. URL: http://php.net/shell_exec.

[9] Cormen Thomas H. et al. *Introduction to Algorithms*. 2nd edition. Cambridge, Massachusetts, USA: MIT Press and McGraw-Hill, 2001. ISBN: 0-262-03293-7.

[10] Tomáš Hlava. *Fáze a úrovně provádění testů*. [Online] Naposledy navštíveno 28. 4. 2015. 2011. URL: http://testovanisoftwaru.cz/tag/akceptacni-testovani/.

[11] Trdlička Jan. "Procesy a vlákna (nepublikovaná přednáška)".

[12] Tomáš Kadlec and Jirkovský Vojtěch. "Symfony - autentizace, autorizace". Přednáška [Online] Naposledy navštíveno 14. 4. 2015. zimní semestr 2014. URL: https://edux.fit.cvut.cz/courses/BI-WT1/lectures/rad/symfony/security.

[13] Sensio Labs. *Twig*. [Online] Naposledy navštíveno 15. 4. 2015. 2015. URL: http://twig.sensiolabs.org/.

[14] Jiří Mlejnek. "Analýza a sběr požadavků". Přednáška [Online] Naposledy navštíveno 24. 3. 2015. letní semestr 2014. URL: https://edux.fit.cvut.cz/courses/BI-SI1/_media/lectures/03/03.prednaska.pdf.

[15] Jiří Mlejnek. "Analýza problémové domény". Přednáška [Online] Naposledy navštíveno 24. 3. 2015. letní semestr 2014. URL: https://edux.fit.cvut.cz/courses/BI-SI1/_media/lectures/04/04.prednaska.pdf.

[16] Petr Olšák. *Úvod do algebry, zejména lineární*. Praha: FEL ČVUT, 2007. ISBN: 978-80-01-03775-1.

[17] Assistant Secretary for Public Affairs. *Wireframing*. [Online] Naposledy navštíveno 12. 4. 2015. 2013. URL: http://www.usability.gov/how-to-and-tools/methods/wireframing.html.

[18] Y. Saad. *Iterative Methods for Sparse Linear Systems*. 2nd edition. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 2003. ISBN: 0898715342.

[19] Jan Schmidt. "Testování použitelnosti". Přednáška [Online] Naposledy navštíveno 28. 4. 2015. letní semestr 2015. URL: https://edux.fit.cvut.cz/courses/BI-TUR/_media/lectures/08/tur8usability.pdf.

[20] SensioLabs. *Symfony*. [Online] Naposledy navštíveno 14. 4. 2015. 2015. URL: http://symfony.com/what-is-symfony.

[21] SensioLabs. *Symfony - The Process Component*. [Online] Naposledy navštíveno 28. 4. 2015. 2015. URL: http://symfony.com/doc/current/components/process.html.

[22] Doctrine Team. *Doctrine*. [Online] Naposledy navštíveno 14. 4. 2015. 2014. URL: http://www.doctrine-project.org/about.html.

[23] Ivan Šimeček. "Technologie OpenMP". Přednáška [Online] Naposledy navštíveno 7. 5. 2015. 2014. URL: https://edux.fit.cvut.cz/courses/BI-EIA/_media/lectures/omp.pdf.

Příloha **A**

Seznam použitých zkratek

GUI Graphical user interface

CRS Compressed Row Storage

CSR Compressed Sparse Row

CSS Cascading Style Sheets

PHP Hypertext Preprocessor

HTML HyperText Markup Language

SVG Scalable Vector Graphics

HTML Vector Markup Language

PID Process Identifier

Příloha **B**

Obsah přiloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `analysis` ..... analýza aplikace
- `design` ..... návrh aplikace
- `samples` ..... screenshoty výsledné aplikace
- `src`
  - `Symfony` ..... zdrojové kódy samotné aplikace
  - `testing` ..... zdrojové kódy testování algoritmů a výsledky
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
    - `img` ..... obrázky použité v práci
  - `install.txt` ..... instalační příručka
- `text`
  - `BP_Waage_Herbert_2015.pdf` ..... text práce ve formátu PDF