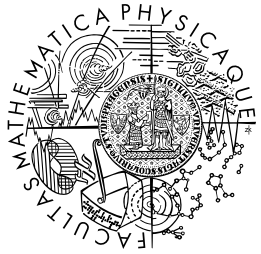


MATEMATICKO-FYZIKÁLNÍ
FAKULTA
Univerzita Karlova

BAKALÁŘSKÁ PRÁCE

Michaela Bobeničová

Vizualizácia výsledkov SMT riešičov

Katedra distribuovaných a spolehlivých systémů

Vedoucí bakalářské práce: doc. RNDr. Jan Kofroň, Ph.D.
Studijní program: Informatika
Studijní obor: Systémové programování

Praha 2023

Prohlašuji, že jsem tuto bakalářskou práci vypracoval(a) samostatně a výhradně s použitím citovaných pramenů, literatury a dalších odborných zdrojů. Tato práce nebyla využita k získání jiného nebo stejného titulu.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona v platném znění, zejména skutečnost, že Univerzita Karlova má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle §60 odst. 1 autorského zákona.

V ..... dne ....

Podpis autora

Ďakujem mojim vedúcim doc. RNDr. Jánovi Kofroňovi, Ph.D. a Mgr. Martinovi Blichovi, Ph.D. za ochotu, čas a pomoc pri tvorbe tejto práce. Taktiež ďakujem mojej rodine a priateľom za trpezlivosť a porozumenie, vďaka ktorým som prácu mohla dokončiť.

Název práce: Vizualizácia výsledkov SMT riešičov

Autor: Michaela Bobeničová

Katedra: Katedra distribuovaných a spolehlivých systémů

Vedoucí bakalářské práce: doc. RNDr. Jan Kofroň, Ph.D., Katedra distribuovaných a spolehlivých systémů

Abstrakt: V súčasnej dobe sa SMT riešiče používajú na riešenie rôznych úloh z mnohých oblastí. Má teda zmysel optimalizovať ich rýchlosť v závislosti od veľkosti a typu problémov. Avšak prechádzať a porovnávať tabuľky s výsledkami behov riešičov je nepraktické. Úlohou tejto práce je vytvoriť grafické prostredie vo forme webovej aplikácie na analýzu výkonu SMT riešičov. Cieľom je umožniť vývojárovi riešiča porovnať výkon viacerých riešičov na konkrétnych voliteľných problémoch. Grafické prostredie obsahuje vizualizácie výkonu v interaktívnych grafoch a tabuľkách.

Klíčová slova: SMT vizualizácia riešiče porovnanie webová aplikácia

Title: Visualization of SMT solvers results

Author: Michaela Bobeničová

Department: Department of Distributed and Dependable Systems

Supervisor: doc. RNDr. Jan Kofroň, Ph.D., Department of Distributed and Dependable Systems

Abstract: Nowadays, SMT solvers are used for solving various problems in multiple fields. Therefore, it makes sense to optimize their speed depending on the size and type of the problems. However, going over and comparing results in tables containing information of the runs of the solvers is impractical. The goal of this thesis is creating a graphical interface for simplifying the analysis of the performance of the solvers. It has a form of a web application. It allows a solver developer to compare performance of multiple solvers on given problem sets. The interface contains visualizations of the performance in the form of interactive plots and tables.

Keywords: SMT visualization solvers comparison web application

Obsah

Kapitola 1

Úvod

1.1 Porovnávanie výkonu

Pri tvorbe nového produktu je porovnávanie výkonu dôležité. V prvom rade, podľa toho zistíme, či sa posúvame správnym smerom, či naozaj vylepšujeme produkt oproti existujúcim variantám. V druhom rade vieme odhadnúť konkurencie-schopnosť produktu.

Avšak ako porovnávať výkon? Musíme si ujasniť otázky, na ktoré chceme dostať odpovede, ktoré nám pomôžu vytvoriť si predstavu o posune. Musíme určiť, ktoré parametre sú pre nás dôležité a relevantné pre daný typ produktu.

Následne budeme potrebovať program na analýzu získaných odpovedí — hodnôt parametrov. To je úlohou tejto práce pre množinu produktov nazývaných SMT riešiče.

1.2 SMT riešiče

Satisfiability Modulo Theories (SMT)[1] je problém o zisťovaní splniteľnosti logických formulí prvého stupňa nad danými teóriami. SMT riešiče (*solvery*) sú programy, ktoré overujú splniteľnosť týchto formulí.

Riešič dostane teóriu a formulu, ktorej splniteľnosť má v danej teórii zistiť. Má zistiť, či existuje ohodnotenie premenných vo formuli také, že celá formula je pravdivá v danej teórii — také ohodnotenie sa nazýva model. Ak zistí, že žiaden model neexistuje, formula je nesplniteľná. Výstupom riešičov bývajú ako výsledok:

- splniteľné — `sat` — a príklad modelu

[1] Hrubý preklad: splniteľnosť vzhľadom na teórie

- nesplniteľné — `unsat` — a dôkaz v danej teórii
- neurčené — `indet` — v prípade, že riešič nenájde odpoveď v danom časovom/pamäťovom limite.

V súčasnej dobe sa SMT riešiče používajú na riešenie rôznych úloh z mnohých oblastí, napríklad ako nástroje pri overovaní modelov, na automatizované dokazovanie tvrdení, pri plánovaní alebo na formálnu verifikáciu programov. Ich úlohou je riešiť matematicky ťažké problémy na obrovských vstupoch. Má teda zmysel optimalizovať ich rýchlosť alebo spotrebu pamäte v závislosti od veľkosti a typu problémov.

Implementácia SMT riešičov je významná pre dnešný svet, a preto sa organizuje súťaž medzi riešičmi SMT-COMP [1]. Popri inom má motivovať vývojárov v pokračovaní na práci s riešičmi. Tá nie je len o samotnej implementácii, ale aj o optimalizáciách — rozvíjaní teórie za tým, hľadať vylepšenia a nové spôsoby prístupu. Tá používa štandardizované problémy — *benchmarky* — a s nimi i štandardizované formáty vstupov a výstupov zahrnuté v projekte SMT-LIB [2]. Pomocou nich vývojári môžu porovnávať riešiče počas vývoja.

Vo všeobecnosti vývojári SMT riešičov majú za cieľ implementovať riešič, ktorý produkuje správne výsledky. V druhom rade sa zameriavajú na optimalizáciu. Pri tomto procese môžu chcieť porovnávať starú verziu riešiča s novou, alebo aj s cudzími ostatnými riešičmi.

SMT riešiče sú zložité nástroje programované v rôznych jazykoch. Ich vývoj je komplexný a na overenie ich funkčnosti sa používajú veľké sady benchmarkov. Preto je porovnanie ich výkonu a výstupov netriviálne, treba to vývojárom zjednodušiť.

Avšak prechádzať a porovnávať tabuľky s výsledkami behov jednotlivých riešičov je pre vývojárov značne náročné. Je to príliš veľa dát na spracovanie človekom. Táto práca vznikla na podnet vývojára riešičov, ktorý chce vizualizovať, prehľadne zobraziť výsledky rôznych riešičov na jednoduché porovnávanie ich výkonu.

1.3 Cieľ práce

Úlohou je vytvoriť aplikáciu na porovnávanie výsledkov riešičov. Cieľom je umožniť vývojárovi riešiča jednoducho porovnať výkon viacerých verzií riešiča na konkrétnych voliteľných problémoch, prípadne svoj riešič porovnať s konkurenčnými riešičmi. Aplikácia by mala mať dostatočne variabilné prostredie na rôzne ovplyvnenie porovnávaných údajov.

1.4 Štruktúra textu

Najprv v kapitole 2 *Analýza problému* vysvetlíme detaily zadania a návrh riešenia. V časti 2.2.1 *Štruktúra vstupných súborov* popíšeme očakávaný formát vstupných dát pre aplikáciu. Následne v kapitole 3 *Technológie* popíšeme použité technológie. V kapitole 4 *Implementácia aplikácie* ako hlavnú časť práce popíšeme architektúru a detaily implementácie aplikácie. Na záver v kapitole 5 *Záver* urobíme zhodnotenie tejto aplikácie. V prílohe C *Užívateľská dokumentácia* doložíme užívateľskú dokumentáciu - príručku.

Kapitola 2

Analýza problému

Úlohou tejto práce je vytvoriť aplikáciu na jednoduché porovnávanie výkonu riešičov. Pre nás ľudí je najjednoduchšia forma spracovania väčšieho množstva dát prostredníctvom obrázkov. Využijeme pri tom rôzne typy grafov. Chceli by sme, aby naša aplikácia nebola závislá na operačnom systéme a aby bola ľahko dostupná pre vývojára riešiča. S ohľadom na to sme zvolili formu webovej aplikácie. Oproti desktopovej aplikácii má výhodu najmä v jednoduchom prístupe prostredníctvom internetového prehliadača.

Ďalšou otázkou je, ako získať dáta zobrazované vo vizualizáciách. Môžu sa spracúvať na strane klienta v prehliadači. To by však musel vývojár zas a znova uploadovať i staré výsledky cudzích riešičov a starších verzií svojich. Budeme potrebovať perzistentné úložisko. Dáta budeme ukladať na serveri, kde ich budeme i spracúvať, aby sa tie množstvá nemuseli presúvať po sieti.

Máme na výber rôzne spôsoby ukaldania, či už v súboroch alebo v databáze. Vzhľadom k tomu, že budeme chcieť rôzne interagovať s dátami a filtrovať ich, databáza je pre nás vhodným nástrojom.

Zhrnutie

Detailnejšou úlohou tejto práce teda je vytvoriť webovú aplikáciu, ktorá sprostredkuje rozhranie do databázy na pridávanie a odoberanie záznamov. Záznamy z databázy sa následne budú zobrazovať v prehľadných vizualizáciách. Keďže rôzne sady benchmarkov sa sústreďujú na rôzne typy problémov, budeme chcieť mať za možnosť zvoliť zobrazovanú benchmarkovú sadu a benchmarky. Budeme chcieť mať aj možnosť samostatne si zobraziť len niektoré výsledky, napríklad kategóriu splniteľné, keďže vrámci kategórií výsledkov môžeme tiež pozorovať vzory.

Čo by sme potrebovali od vizualizácií? Chceli by sme vidieť prehľadné všeobecné porovnanie viacerých riešičov naraz, aby sme získali hrubú pred-

stavu o rozdieloch vo výkone. Ku tomu by sme potrebovali vidieť detailnejšie porovnanie riešičov - v závislosti od riešených benchmarkov. Tieto dva typy sa tým pádom budú príjemne dopĺňať. Popri tom by sme chceli ešte vidieť nejaké prehľadné súhrnné štatistiky pre riešiče v závislosti od sád na tabuľkové porovnanie.

Nepodarilo sa nám nájsť žiadne vhodné už existujúce aplikácie na vizualizáciu porovnania výkonu riešičov. Aplikácia bola teda vytvorená podľa požiadaviek vývojára riešičov.

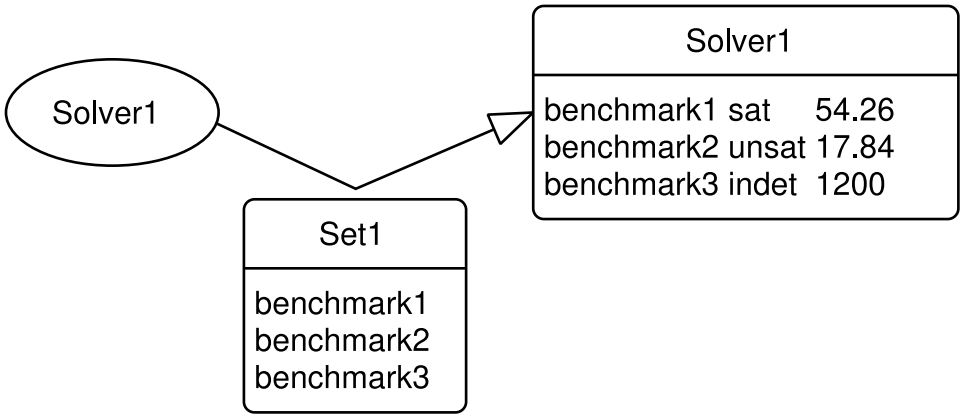
2.1 Použitie aplikácie

Ako vyzerá bežné použitie tejto aplikácie? Príkladom takej situácie nech je vývojár riešiča, ktorý chce porovnať aktuálnu verziu svojho riešiča s predošlými a s vybranými konkurenčnými riešičmi:

1. Samostatne si zoženie benchmarky, na ktorých spustí tieto riešiče.
2. Výsledky benchmarkov uloží a naformátuje do nami požadovaného formátu.
3. Otvorí webovú aplikáciu a v nej databázové rozhranie.
4. Rozhranie bude zabezpečené jedným heslom. Vývojár zadá toto heslo a následne uploadne vytvorené súbory.
5. V súhrnnej grafovej vizualizácii na prvý pohľad jasne uvidí, ktorý riešič bol výkonnejší.
6. Pre detailnejšie určenie rozdielov si zobrazí vizualizácie, kde bude zobrazený výkon pre jednotlivé benchmarky v porovnaní s inými riešičmi.
7. Môže si zvoliť, že si pozrie len výsledky behov do x sekúnd alebo len výsledky niektorých benchmarkov.
8. Samostatne zhodnotí, aký efekt mali jeho zmeny na výkon riešiča, a podľa toho pokračuje vo vývoji a úpravách.

2.2 Formát vstupu

Od užívateľa očakávame súbory, ktoré aplikácia spracuje na vstup pre databázu. Predpokladáme, že existuje viacero sád (*setov*) benchmarkov. Každá



Obr. 2.1 Riešič spustíme na jednej celej sade benchmarkov. Výsledky zapíšeme do jedného súboru s názvom riešiča ako názvom súboru, s možnou príponou ktorá sa odignoruje. Súbor má obsahovať názov benchmarku, odpoveď, ktorú vrátil riešič, a čas, koľko sekúnd riešiču trvalo vyriešiť daný benchmark.

Výpis kódu 1 Ukážka vstupného súboru s výstupom riešiča. Prvý stĺpec obsahuje názvy benchmarkov, druhý výsledok riešiča a tretí čas, koľko mu trvalo vyriešiť daný problém. Časové obmedzenie bolo nastavené na približne 1200 sekúnd.

```
TM/p4-driverlogNumeric_s8 sat 0.26
TM/p5-zenonumeric_s5 sat 0.19
TM/p-1-bucket_s9 sat 0.40
TM/p3-driverlogNumeric_s8 sat 0.21
TM/p6-zenonumeric_s5 sat 0.19
TM/p-3-bucket_s10 sat 1.56
TM/p7-driverlogNumeric_s7 sat 1.23
TM/p5-driverlogNumeric_s9 unsat 0.24
TM/p2-driverlogNumeric_s10 sat 0.20
tropical-matrix/constraint-343414 indet 1203.74
tropical-matrix/constraint-1007893 indet 1203.33
tropical-matrix/constraint-330061 sat 216.90
tropical-matrix/constraint-223827 sat 8.86
tropical-matrix/constraint-436218 sat 225.03
tropical-matrix/constraint-269735 sat 178.50
tropical-matrix/constraint-251380 sat 274.60
tropical-matrix/constraint-556101 indet 1204.04
tropical-matrix/constraint-413064 indet 1203.43
tropical-matrix/constraint-199552 sat 50.78
```

sada je samostatná časť, nezávislá na ostatných. Užívateľ si samostatne nájde benchmarky, ktoré vytvoria sadu alebo viac sád, a spustí na nich porovnávané riešiče. Pre jednu sadu a jeden riešič vytvorí jeden súbor obsahujúci všetky benchmarky tejto sady. Pre lepšie pochopenie viď obrázok 2.1.

2.2.1 Štruktúra vstupných súborov

Súbor má obsahovať názov benchmarku, odpoveď, ktorú vrátil riešič, a hodnotu, napríklad čas, koľko sekúnd riešiču trvalo vyriešiť daný benchmark. Možné odpovede riešiča momentálne sú `sat`, `unsat` a `indet`. Názov súboru by mal jasne označovať, ktorý riešič vytvoril tieto výsledky. Ten, mimo prípony, bude použitý ako názov riešiča vo vizualizáciách.

Štruktúra vstupného súboru vyzerá nasledovne:

- Názov riešiča — názov súboru bez prípony
- Názov benchmarku — prvá informácia v riadku výsledku
- Výsledok riešiča — druhá informácia v riadku výsledku
  - `sat` — vstupná formula je splniteľná v danej teórii
  - `unsat` — pre vstupnú formulu neexistuje model v danej teórii
  - `indet` — riešič nezvládol vyriešiť tento problém v ponúknutom čase
- Hodnota meranej veličiny — tretia informácia v riadku výsledku.

Pre ukážku vstupného súboru viď výpis 1.

Rôzne sady benchmarkov sa sústreďujú na rôzne problémy, a teda majú i rozdielnu zložitosť a veľkosť problémov. Preto očakávame veľký rozptyl hodnôt neranej veličiny, napríklad čas na vyriešenie.

2.3 Vizualizácie

Možný spôsob ako vizualizáciami porovnať výkon riešičov je pomocou rôznych grafov a tabuliek. V ďalších sekciách tejto kapitoly nasleduje podrobnejší popis týchto spôsobov a v ďalších častiach aj ich implementácia, viď kapitola 4.

Na prehľadné všeobecné porovnanie viacerých riešičov naraz sa v SMT kruhoch zvyčajne používa takzvaný **cactus plot**. Pre každý riešič obsahuje jednu krivku znázorňujúcu jeho výkon. Nižšie ho bližšie popíšeme.

Doplňujúce detailnejšie informácie zobrazíme pomocou bodového grafu, takzvane **scatter plot**. Je to graf na porovnanie dvoch riešičov a ich výkonu vzhľadom na benchmarky, čo nám vnesie ďalšie svetlo do fungovania daných riešičov.

Budeme chcieť rôzne obmedzovať zobrazované informácie v grafoch, takže budeme potrebovať i klasické **tabuľkové zobrazenie** s adekvátnou filtrovacou funkcionalitou.

Na záver, hoc to rozhodne nie je posledný možný spôsob vizualizácie, implementujeme aj tabuľku so štatistikami pre rýchle prehľadné porovnanie pomocou čísel — **sumarizačnú tabuľku**.

Keďže vývoj riešičov je úzko spätý aj s akademickými prácami, budú tieto grafy exportovateľné do viacerých vektorových a rastrových formátov.

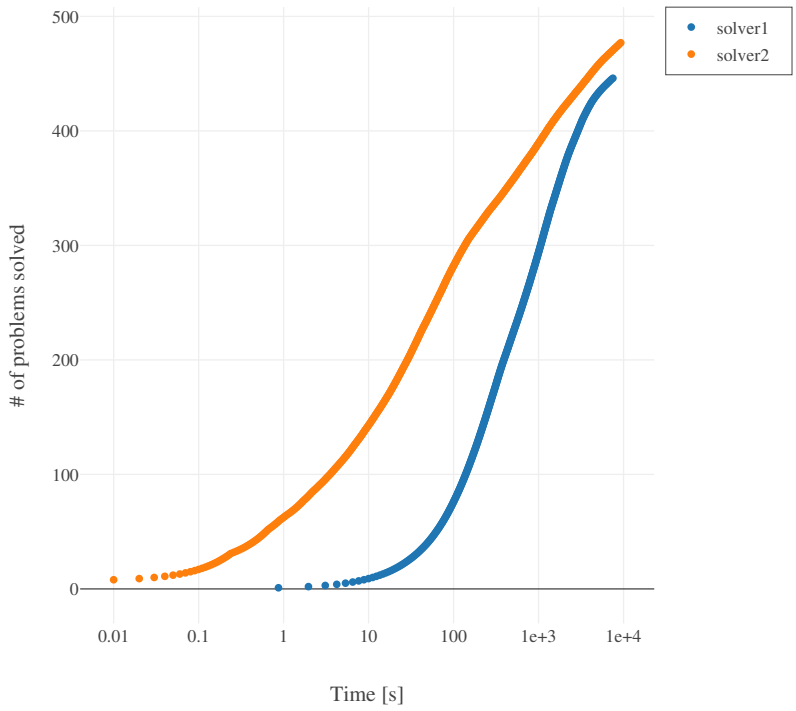
2.4 Grafové vizualizácie

Človek nie je stroj. Ťažko sa mu hľadajú vzory a tvoria závery v tabuľkových dátach. Z grafov však omnoho ľahšie vypozoruje potrebné vzťahy. V tejto aplikácii sme implementovali dva druhy pohľadov na dáta — cactus plot a scatter plot.

2.4.1 Cactus plot

V prostredí SAT a SMT riešičov sa používa na porovnávanie popri inom aj takzvaný cactus plot. Je to graf kumulatívneho času — aký čas trvá riešiču vyriešiť niekoľko problémov. Niekedy sa tiež nazýva *survival plot/function*, čiže *graf/funkcia prežitia*. Funguje nasledovne [3]:

1. *Pre každý riešič samostatne*
   (a) *Vyriešime benchmark b_i zaznačiac si čas behu t_i, do nejakého limitu. (Tento krok nepatrí do úloh našej práce.)*
   (b) *Vzostupne zotriedime časy t_i vynechajúc tie, ktoré prekročili časový limit.*
   (c) *Skonštruujeme body $(t_1, 1)$, $(t_1 + t_2, 2)$, ... $(\sum_{i=1}^{k} t_i, k)$.*
2. *Zobrazíme skonštruované body v grafe, na x-ovej osi nech je čas a na y-ovej počet vyriešených benchmarkov. Rôzne riešiče odlíšime farbami. Môžeme použiť logaritmickú mierku pre čas.*



Obr. 2.2 Cactus plot znázorňujúci koľko benchmarkov vie ktorý riešič vyriešiť za aký čas. V tomto prípade oranžový riešič zvládne vyriešiť viac problémov za kratší čas.

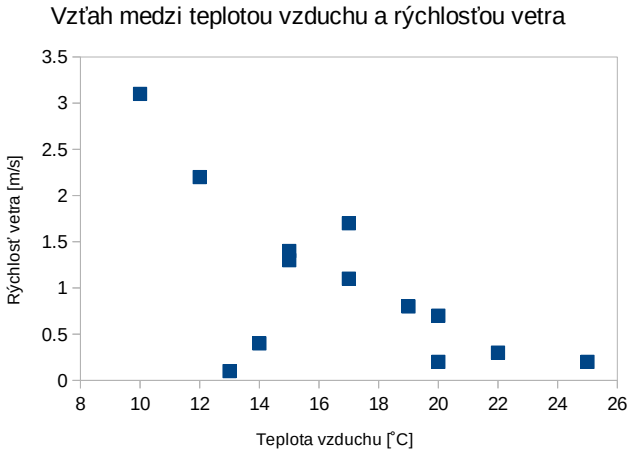
Dosiahneme tým takýto graf, viď obrázok 2.2.

Riešič, ktorého krivka je v nejakom bode času vyššie, je výkonnejší čo sa počtu vyriešených problémov za čas týka. Inými slovami, čím viac je krivka vľavo hore, tým lepšie. Znamená to, že za kratší čas vyriešil riešič viac problémov.

Tento graf prehľadne, avšak nie detailne zobrazuje výkon viacerých riešičov súčasne. Vzhľadom k metodike akou tvoríme body kriviek, nevieme určiť vzťahy výkonov k jednotlivým benchmarkom, keďže body rôznych riešičov môžu referovať na rôzne benchmarky. Tento problém vyriešime ďalším typom grafu — scatter plotom.

2.4.2 Scatter plot

Bodový graf je diagram používajúci karteziánsky systém súradníc na zobrazenie hodnôt pre dve premenné. Dáta sú znázornené ako množina bodov,



Obr. 2.3 Ilustračný scatter plot s ilustračnými dátami. Zo scatter plotov môžeme vyčítať prípadný vzťah medzi dvoma premennými.

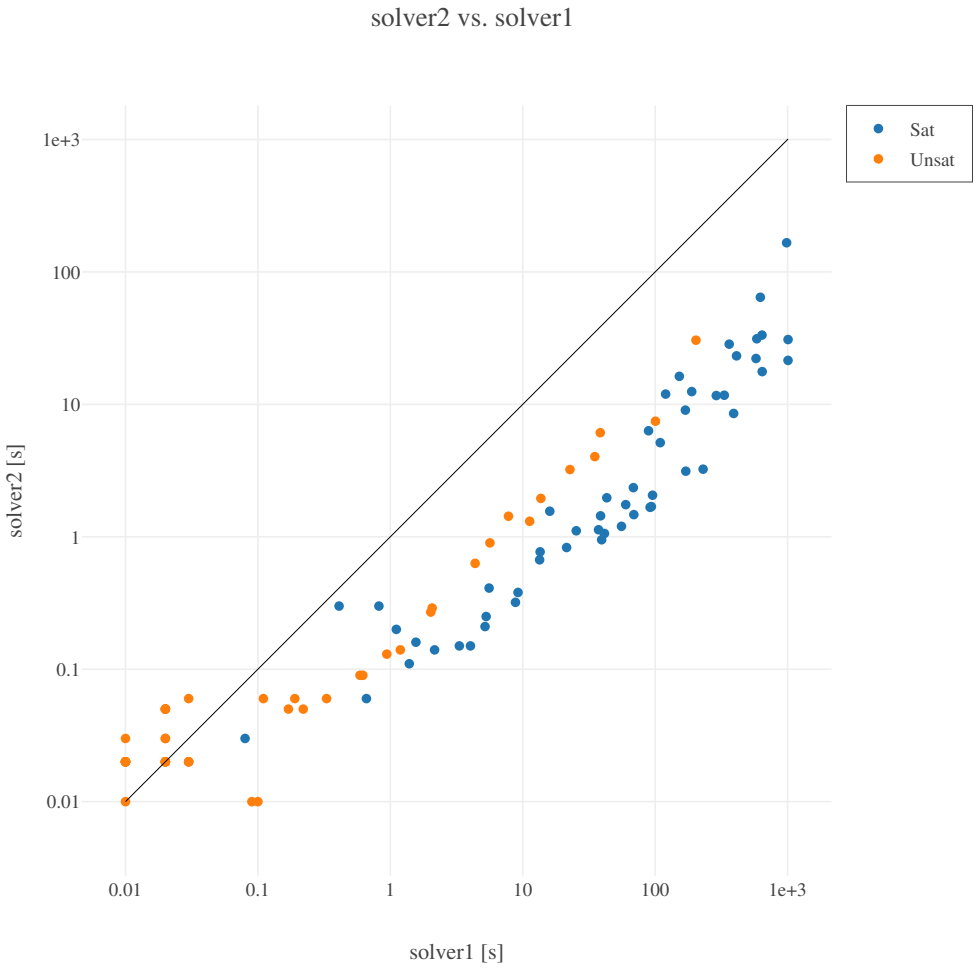
kde každý bod znázorňuje hodoty premenných. Pri použití rôznych označení bodov, napríklad rôzne farby, tvary alebo veľkosti, vieme vyjadriť aj ďalšie premenné. Viď obrázok 2.3.

Pomocou tohto grafu môžeme jednoducho vyčítať vzťah medzi danými premennými.

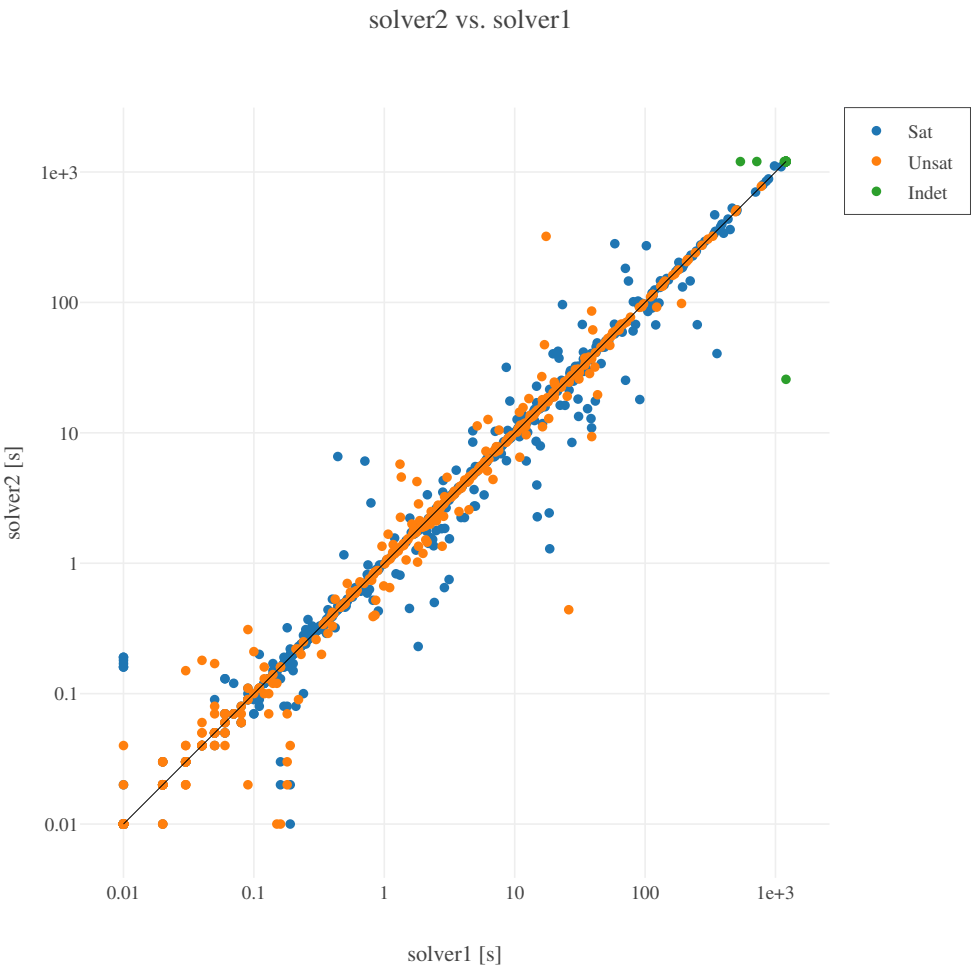
Porovnanie riešičov

V našom prípade tento typ grafu slúži na porovnávanie riešičov spúšťaných na rovnakej sade benchmarkov. Každý bod v grafe reprezentuje jeden benchmark, kde jeho x-ová súradnica reprezentuje hodnotu pre jeden riešič a y-ová súradnica pre druhý. Body na diagonále (t.j. $x = y$) vyjadrujú, že oba riešiče dosiahli rovnakú hodnotu a body nad/pod diagonálou, že riešič na x/y-ovej osi dosiahol nižšiu hodnotu, viď [4, sekcia 7.6]. Zobrazovaná hodnota je zvyčajne čas vyriešenia daného benchmarku, spotreba pamäti pri riešení daného benchmarku, atď.

Každý bod má i farbu reprezentujúcu výsledok behu. Ak sa výsledky oboch riešičov pre tento benchmark zhodujú, majú farbu podľa legendy. Ak je aspoň jeden z nich hodnoty `indet`, celý bod bude mať farbu podľa tejto hodnoty. A ak niekedy nastane prípad, že sa hodnoty nebudú zhodovať — čo by sa pri správnom fungovaní riešičov nemalo stať — ten bod bude zobrazený ďalšou farbou podľa legendy. Napriek tomuto zobrazeniu tento graf nie je vhodný na vizualizáciu vlastného výsledku riešiča, keďže sa body prekrývajú a v prípade vypršania časového limitu sa z farby nedozvieme výsledok druhého riešiča. Viď obrázok 2.4 a obrázok 2.5.



Obr. 2.4 Scatter plot zobrazujúci porovnanie výkonu dvoch odlišných riešičov. Každý bod reprezentuje čas, koľko trvalo vyriešiť konkrétny benchmark. Čím bližšie sú body ku pomocnej diagonálnej línii, tým majú dané riešiče podobnejší výkon. V tomto prípade je väčšina bodov pod diagonálou, čo znamená, že riešiču 1 trvalo dlhšie vyriešiť tieto benchmarky.



Obr. 2.5 Scatter plot zobrazujúci porovnanie výkonu dvoch podobných riešičov. Väčšina bodov je blízko diagonály, čo znamená, že tieto riešiče sú výkonovo veľmi podobné.

2.4.3 Cactus plot vs. scatter plot

Môžeme sa na to dívať tak, že cactus plot je vysokoúrovňová vizualizácia výkonu riešičov, ideálna na rýchle hrubé porovnanie. Môžeme z neho vyčítať všeobecnú výkonnosť relatívne voči iným riešičom. Avšak viacero informácií sa v ňom môže stratiť.

Scatter plot, na druhú stranu, môže byť ako nízkoúrovňová forma vizualizácie. Zobrazuje porovnanie výkonu vzhľadom na jednotlivé benchmarky. V tomto zobrazení môže vývojár nájsť vzťahy medzi výkonom a typom problémov alebo výsledkov.

2.5 Tabuľkové vizualizácie

Ako sme spomínali vyššie, grafy sú príjemnejšie na vypozorovanie vzťahov medzi dátami. Ak však ide o detaily, presnosť tabuľkových zobrazení nemá konkurenciu. Tabuľky sa nám zídu aj na manipuláciu s dátami v databáze a pri určovaní, ktoré dáta vôbec chceme zobraziť v grafoch.

2.5.1 Tabuľkové zobrazenie

Scrollovanie dlhým zoznamom nemusí byť príjemné, a teda sme implementovali tabuľku ako stránkovanú. Dáta v tabuľke rozdelíme na diely o veľkosti n podľa zadania užívateľa — to budú naše strany. Ku tabuľke zobrazíme tlačidlá na prepínanie medzi týmito stranami. Napriek tomu, že vždy bude zobrazená len jedna strana, interne sa bude zobrazovať v grafoch a pri operáciách úpravy databázy brať do úvahy celá tabuľka, nie len zobrazená časť.

Pri detailnom prezeraní výsledkov riešičov treba výsledky nejakým spôsobom obmedziť, keďže prechádzať takým množstvom dát by nebolo ideálne. Keďže ju budeme využívať i na určovanie, ktoré dáta zobraziť v grafoch, aj pri tom potrebujeme funkcionalitu filtrovania. Môžeme chcieť filtrovať podmnožiny benchmarkov, podmnožiny výsledkov alebo intervaly veličiny. V databáze bude mať každý stĺpec svoj typ. Podľa typu stĺpca implementujeme i typ filtrovania — podrobne to bude rozobraté v sekcii 4.5.

Na prezeranie tabuľkových výsledkov sa môže hodiť aj funkcionalita triedenia, minimálne kvôli zotriedeniu veličín a porovnaniu, ktoré benchmarky boli zvládnuté lepšie.

2.5.2 Sumarizačná tabuľka

Vrámci každej sady benchmarkov, ktoré sa vyznačujú rôznymi vlastnosťami, je možné zobraziť i štatistiky. Sú to informácie o súhrnnej úspešnosti jednotlivých

riešičov. V tabuľke sú nasledujúce údaje:

- Počet vyriešených benchmarkov
- Percento vyriešených benchmarkov
- Počet `sat` výsledkov
- Počet `unsat` výsledkov
- Počet benchmarkov, ktoré vyriešil daný riešič ako jediný
- Mená jedinečne vyriešených benchmarkov

Kapitola 3

Technológie

Chceme, aby táto práca nebola závislá na operačnom systéme, ale ladená bola len na Linuxe. Podľa toho sme vyberali použité knižnice a frameworky.

3.1 Back-end

Projekt je písaný v prostredí *.NET 7.0* v jazyku *C# 11* [5]. C# je silno typovaný. Založený je na objektovo orientovaných princípoch, ale inšpirovaný i mnohými ďalšími, ako napríklad funkcionálne programovanie [6]. Patrí medzi najznámejšie jazyky medzi programátormi. Jeho ekosystém je rozsiahly - nie len že má rozsiahlu štandardnú knižnicu, ale obsahuje aj množstvo open-source knižníc od tretích strán. V samotnom jazyku je taktiež možnosť používať *LINQ* [7] (Language-Integrated Query) na komunikáciu s databázami, čo je pre nás príjemným bonusom.

3.2 Databáza a interakcia

Ako databázu sme zvolili *SQLite* [8]. Je open-source, zadarmo a jednoduchá. S tým súvisí i použitie *Entity Framework Core* [9] technológie pre SQLite, ktorá tiež patrí do .NET ekosystému. Je to mapovač medzi objektami a databázami, ktorý komunikuje s databázou i čo sa týka zmien v dátach, má na starosti aj updatovanie.

3.3 Front-end

Našou úlohou je vytvoriť webovú aplikáciu. V .NET nájdeme framework na to určený - *Blazor Server* [10]. Používa sa v ňom C#. Blazor Server

framework, na rozdiel od Blazor WebAssembly frameworku, beží čisto na serveri, a klientovskému internetovému prehliadaču posiela už hotovú stránku. Pri nejakej zmene v užívateľskom rozhraní na serveri vykoná zmenu a nájde minimálny spôsob, ako dosiahnuť danú zmenu, a tento spôsob pošle naspäť klientovi, kde sa časť stránky prekreslí [11]. Práve takýto prístup sa nám hodí, lebo všetky dáta sú na serveri, kde sa spracujú a vytvoria potrebné grafy a tabuľky. Naša aplikácia už len sprostredkuje vizualizácie a rozhranie.

3.4 Funkcie aplikácie

3.4.1 Grafy

Jeden zo spôsobov vizualizácie sú grafy. Vo svete JavaScriptu je známa knižnica Plotly.js [12]. My využijeme knižnicu, ktorá je obálkou tejto - *Plotly.Blazor* [13]. Plotly je knižnica so širokým spektrom interaktívnych grafov, ktoré sú vysoko upraviteľné podľa našich požiadaviek. Plotly.Blazor funguje ako komponent vrámci Blazor-a. Mimo nejakých naozaj špecifických vecí sa JavaScriptu priamo ani nedotkneme.

3.4.2 Export grafov

Ďalšia z funkcionalít našej aplikácie je možnosť exportovať grafy do rôznych vektorových i rastrových obrázkových formátov. O väčšinu z nich sa stará práve knižnica Plotly.js. Avšak tá neponúka možnosť exportu do formátu PDF, ktorú sme implementovali samostatne.

Ponúkame nasledujúce možné formáty na export: svg, pdf, png, jpeg a webp.

3.4.3 Export do PDF

PDF formát sme chceli využiť ako vektorovú variantu. Ako základ pre PDF obrázok sme zvolili SVG výstup z Plotly.js. Hľadali sme vhodnú knižnicu, ktorá by bola schopná vytvoriť PDF obrázok z SVG súboru.

Najprv sme skúšali knižnicu Magick.NET [14]. Je to C# wrapper pre nástroj ImageMagick a nevyžaduje inštaláciu tohto nástroja na serveri. To nám prišlo vhodné, lebo sa snažíme o aplikáciu nezávislú na operačnom systéme a toto by bola ďalšia závislosť navyše na vyriešenie. Nanešťastie, táto knižnica nemá dostatočnú dokumentáciu a nepodarilo sa nám implementovať vektorovú konverziu SVG obrázka na PDF.

Následne sme skúšali externý nástroj CairoSVG [15]. Je to wrapper pre nástroj cairo implementovaný v jazyku Python. Ten sa nám už podarilo použiť, avšak niektoré prvky grafov nezobrazoval správne. Taktiež je tu problém spomínaný už vyššie — bola by to ďalšia závislosť.

Túto funkcionalitu sme nakoniec implementovali pomocou knižnice *iText 7* [16]. Je open-source, pre nekomerčné využitie zadarmo, a jednoducho sa používa. Taktiež je stále vyvíjaná. Má i možnosť vytvárať PDF/A súbory, avšak to by vyžadovalo väčšiu časovú investíciu. Bolo by treba do hĺbky pochopiť pochopiť fungovanie formátu PDF. Môže to byť jedna z možností na rozvitie v budúcnosti.

Kapitola 4

Implementácia aplikácie

Štýl implementácie tejto aplikácie je ovplyvnený použitým frameworkom Blazor. Ten používa razor komponenty na vytváranie častí webových stránok. Tie sa môžu do seba vnorovať a následne sa použijú v tzv. pages, t.j. triedach reprezentujúcich jednotlivé stránky daného webu.

V tejto kapitole bližšie popíšeme niektoré systémy a princípy.

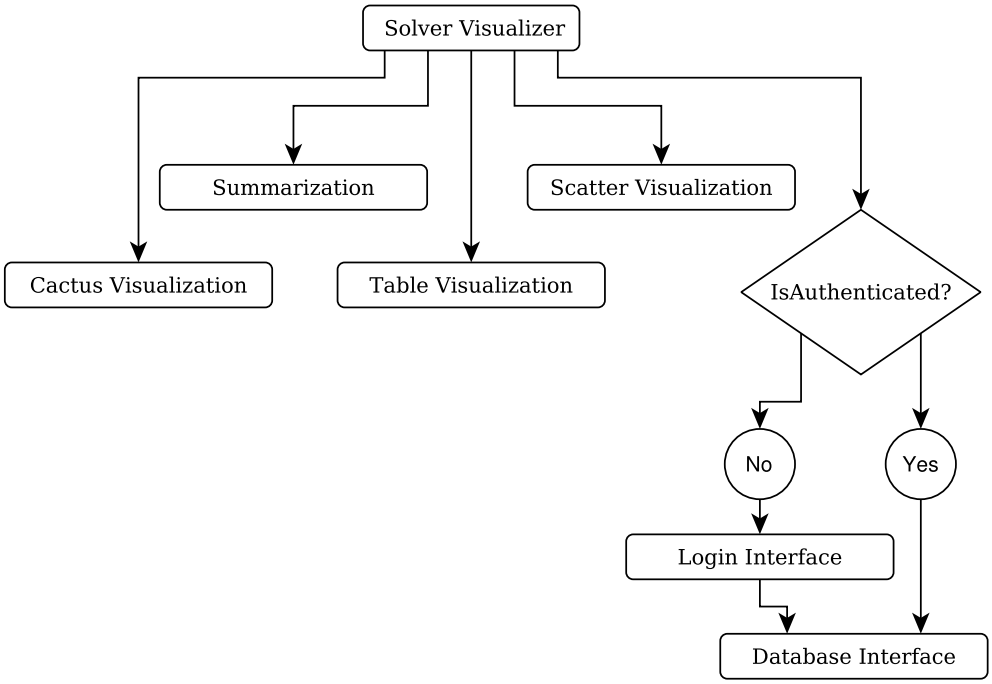
4.1 Štruktúra

Stránky rozhraní sú implementované samostatne, viď obrázok 4.1 a sekciu 4.7, avšak môžu používať rovnaké komponenty. Každá trieda má jasne danú svoju úlohu. Môžeme ich rozdeliť do skupín podľa formy:

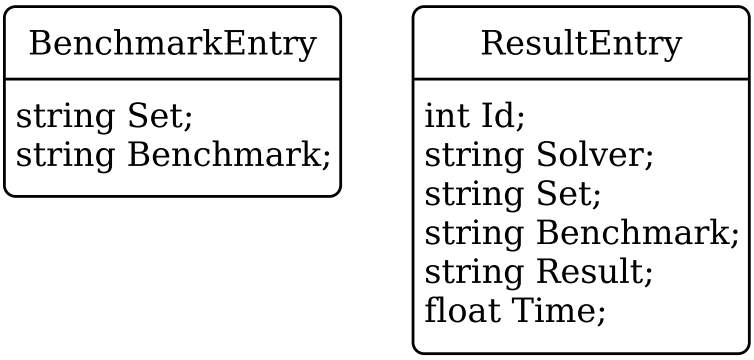
- **Stránka**
  Routovateľný komponent reprezentujúci obsah zobrazenej stránky.
- **Komponent**
  Zobraziteľný objekt s konkrétnou úlohou.
- **Služba**
  Trieda sprostredkúvajúca informácie alebo všeobecne službu.
- **Pomocná trieda**
  Vedľajšia trieda pre lepšiu prácu a orientovanie sa v programe.

4.2 Interná reprezentácia dát

Interne dáta reprezentujeme dvojako. Viď obrázok 4.2. V prvom rade, ak užívateľ uploaduje novú sadu, uložíme si do databázy názov sady a jej bench-



Obr. 4.1 Vrámci užívateľského prostredia má užívateľ dostupné tieto rozhrania, stránky.



Obr. 4.2 Dve triedy reprezentujúce vstupné dáta v databáze a v aplikácii. BenchmarkEntry pre ľahšie hľadanie benchmarkov v sade, a ResultEntry s kompletnými údajmi jednotlivých výsledkov.

marky podľa prvého uploadnutého súboru. Názov sady zadáme v grafickom prostredí pri uploadovaní súborov.

Po uložení samotnej sady prejdeme všetky súbory. Ak v nejakom z nich chýba benchmark z jeho sady, daný súbor odignorujeme, ale informujeme užívateľa. Tým zaručíme, že všetky výsledky pre nejakú sadu vždy obsahujú práve tie isté benchmarky.

4.3 Ochrana prístupu

Keďže naša aplikácia nie len zobrazuje dáta zo serveru, ale ich aj sprístupňuje na úpravu, je vhodné upravovacie rozhranie ochrániť heslom.

Na začiatok popíšeme rozdiel medzi autentifikáciou a autorizáciou. Autentifikácia je proces overenia identity užívateľa. Autorizácia je proces overenia, či daný užívateľ má prístup k nejakému objektu [17].

Ochranou prístupu sa zaoberajú triedy z namespace-u Components.Authorization [18]. Z nich využijeme `AuthenticationStateProvider` a `AuthorizeRouteView`.

4.3.1 Autentifikácia

O proces autentifikácie sa stará naša trieda `Authentication`, ktorá je špecializáciou triedy `AuthenticationStateProvider`. Zabezpečuje overenie zadaného hesla, prihlásenie a odhlásenie, a priradenie role k aktuálnemu užívateľovi, resp. jednému sieťovému obvodu, ktorý zabezpečuje komunikáciu medzi klien-

tom a serverom. Následne zabezpečuje aj odhlásenie užívateľa. Sprístupňuje aplikácii informáciu o role aktuálneho užívateľa - či už je anonymný návštevník alebo administrátor[1].

4.3.2 Autorizácia

Momentálne sú užívatelia obmedzení čo sa týka prístupu k databázovému rozhraniu. Pomocou triedy resp. komponentu `AuthorizeRouteView` špecifikujeme čo sa má stať, keď užívateľ nie je autorizovaný interagovať s nejakým objektom. Avšak pre jednotlivé stránky treba označiť, ktoré role tam majú prístup. Na to použijeme triedu `AuthorizeAttribute` [19], ktorá reprezentuje atribút `Authorize`. Tým sme vyriešili ochranu prístupu k databázovému rozhraniu.

4.4 Rozhranie do databázy

Vzhľadom na to, že s databázou interagujeme len pridávaním, odoberaním, filtrovaním a triedením vo fixných formátoch, implementovali sme triedu `DatabaseApi`. Tá sprostredkúva metódy s týmito funkcionalitami. Keďže táto trieda je len obalom rozhrania do databázy a nezávisí na užívateľoch, je použitá ako singleton - všetky spojenia využívajú to isté `DatabaseApi`.

Táto trieda komunikuje s databázou prostredníctvom triedy `DatabaseContext`, ktorá je špecializáciou `DbContext`. Viď obrázok 4.7. Tá reprezentuje jednu sadu operácií nad databázou. Nepodporuje paralelný prístup [20]. Kvôli tomu používame "továreň" kontextov. Každá metóda `DatabaseApi` teda používa iný kontext. Tie sa recyklujú, keď dokončia svoju prácu, čo zastrešuje ich "továreň" [21].

Implementujeme tu proces triedenia, ktorý je bližšie popísaný v sekcii 4.9.2, taktiež aj pridávanie a odoberanie výsledkov a benchmarkov.

Na pridanie sady benchmarkov potrebujeme súbor s výsledkami nejakého riešiča. Z neho si vyberieme názvy benchmarkov a uložíme ich do databázy spolu so zadaným menom sady.

Pridávať výsledky môžeme hromadne, pomocou viacerých súborov. Benchmarky každého riešiča (súboru) porovnáme s benchmarkami, ktoré boli priradené k danej sade. Porovnávame, či tam sú práve tie, nezáležiac na poradí. Ak je v nejakom súbore chyba, jeho obsah sa ani čiastočne neuloží — preskočí sa. Avšak súbory sa ukladajú nezávisle — práve chybné súbory sa preskočia, ostatné sa uložia.

[1]Pod administrátorom sa v tomto kontexte myslí osoba, ktorá môže meniť obsah databázy.

4.5 Filtrovanie

Otázka filtrovania je zložitejšia. Jednoduché filtrovanie pomocou podreťazcov nie je ideálne, keďže už len výsledky `sat` a `unsat` oba spĺňajú vyhľadávanie podreťazca `sat`.

Mohli by sme povedať, že budeme filtrovať podľa prefixov, čím by sme tento problém vyriešili. Tým však vzniká nový problém, a to, že napísať filtrujúci výraz môže byť zdĺhavé a prišli by sme tým o flexibilitu pri filtrovaní.

Zvolili sme teda riešenie, kde filtrovaný výraz je regulárnym výrazom. Viď [22] a PDF informáciu [23] pre informácie ku syntaxi nami používaných regulárnych výrazov.

4.6 Export grafov do PDF

Knižnica Plotly.js, ktorá je základom našich grafov, viď sekcia 4.8.1, nemá možnosť vytvorenia alebo stiahnutia grafu vo formáte PDF.

Funkcionalitu sme implementovali pomocou knižnice iText 7. Knižnica Plotly.js vie vytvoriť SVG obrázok grafu. Ten, pomocou knižnice iText 7, skonvertujeme na PDF. Samotná knižnica iText 7 má funkcionalitu na vytvorenie súborov aj verzie PDF/A, ale to sa nám nepodarilo doimplementovať. Vytvárame teda PDF súbory bez špeciálnych dodatočných požiadaviek.

Keďže vytvárame PDF súbor lokálne na serveri, bude ho treba niekedy zmazať. Avšak nemáme ako vedieť, kedy sa obrázok u klienta dosťahoval. Implementovali sme teda službu `PlotFilesCleanup`, ktorá v konfigurovateľných intervaloch času zmaže tieto pomocné PDF súbory staršie ako daný čas.

4.6.1 Priebeh sťahovania v PDF formáte

Od Plotly.js grafu po stiahnutý PDF súbor u užívateľovi je dlhá cesta:

1. Z grafu vytvoríme v pamäti SVG obrázok grafu volaním metódy z Plotly.js.

2. Vytvoríme jedinečné meno pre vytváraný súbor pomocou číselníka a zámku na ňom, aby sme sa vyhli korupcii dát v prípade paralelného volania pre viacero klientov.

3. Pomocou knižnice iText 7 vytvoríme na serveri PDF súbor s grafom.

4. Pomocou skriptu v súbore `Pages/_Layout.cshtml` nasimulujeme akciu stiahnutia od užívateľa.

4.7 Komponenty

V nasledujúcich sekciách sa sú schémy reprezentujúce grafy závislostí medzi komponentami. Vrcholy grafu s bielym pozadím a plnou čiarou hranice reprezentujú daný komponent alebo stránku v zdrojovom kóde. Avšak komponentov je mnoho, rýchlo by sa stali schémy neprehľadnými. Niektoré vrcholy sme spojili do skupín nazvaných podľa ich najdôležitejších komponentov. Postupne v tejto kapitole budú i schémy s obsahom jednotlivých skupín.

4.8 Grafy

Grafy v našej aplikácii sú postavené na komponente PlotlyChart z knižnice Plotly.Blazor [24]. Oba typy využívajú ako typ grafu scatter plot, hoc zobrazujú sémanticky rozdielne veci.

4.8.1 PlotlyChartImageable

Knižnica Plotly.Blazor neimplementuje možnosť vytvoriť alebo stiahnuť obrázok vo vektorovom formáte napriek tomu, že knižnica pod ňou, Plotly.js, túto možnosť má. Preto sme vytvorili túto triedu dediacu od PlotlyChart a pomocou JavaScript interoperability [25] sme doimplementovali túto funkcionalitu. Prakticky je to volanie JavaScript-ovej metódy knižnice Plotly.js. Viď obrázok 4.3.

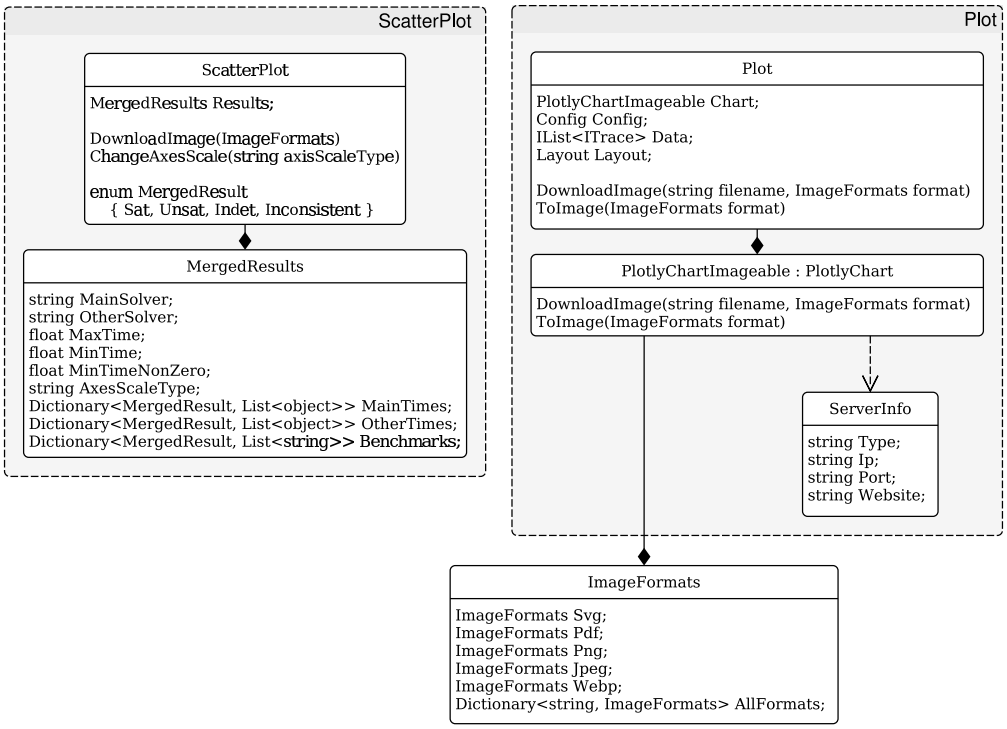
Samotný JavaScript sa nachádza v súbore `Pages/_Layout.cshtml`. Pre viac informácií viď sekcia 4.6.

4.8.2 Plot

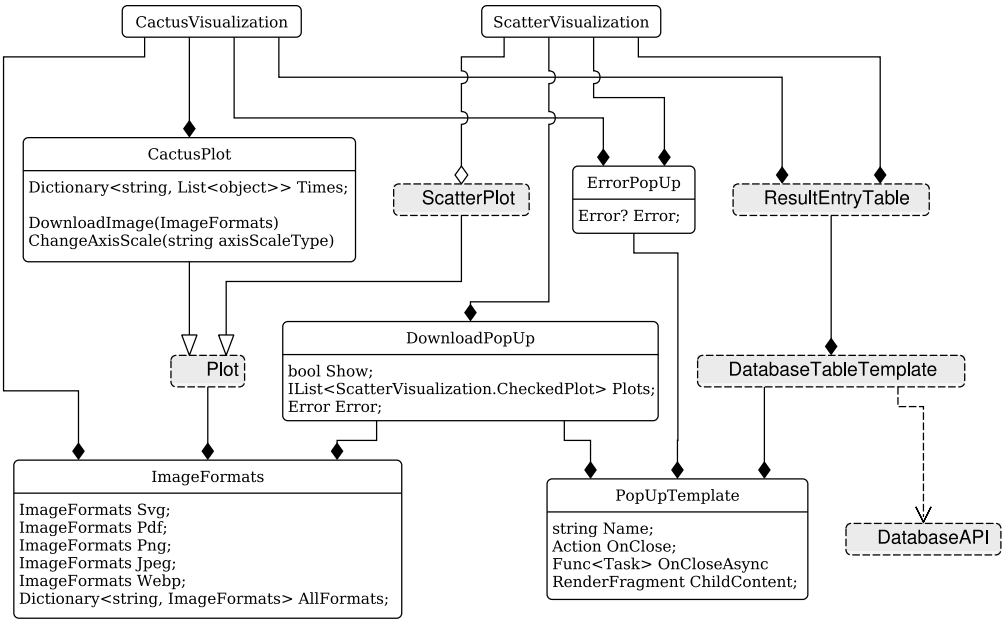
Komponent `PlotlyChartImageable` sme obalili do tejto triedy pre ľahší prístup k jeho properties z pohľadu dediacich tried od triedy `Plot`. Viď obrázok 4.3. Tento medzikomponent je výhodný na prácu s properties PlotlyChartImageable, keďže komponenty by nemali za behu meniť vlastné properties kvôli vlastnostiam fungovania Blazora [26].

4.8.3 CactusPlot a CactusVisualization

Stránka s vizualizáciou pomocou cactus plotu `CactusVisualization` obsahuje tabuľku s výsledkami `ResultEntryTable`, ktorá má funkciu filtrovania, viď sekcia 4.5. Je tam i samotný graf - `CactusPlot`. Možnosť stiahnutia obrázku grafu je implementovaná pomocou tlačidla, ktoré rozvinie časť obsahu



Obr. 4.3 Graf závislostí pre Plot a ScatterPlot.



Obr. 4.4 Graf závislostí pre CactusVisualization a ScatterVisualization.

stránky. Zobrazí sa možnosť na voľbu formátu obrázku typu `ImageFormats`. V prípade, že nastane nejaký problém, zobrazí sa vyskakovacie okno s chybovou hláškou vraviac čo je problém. Viď obrázok 4.4.

Príprava dát

Trieda `CactusVisualization` spracúva dáta z databázy do vhodného formátu pre `CactusPlot` podľa postupu zo sekcie 2.4.1. Vytvorí slovník, kde kľúčom je riešič a hodnotou je list objektov, čo sú pretypované kumulatívne časy typu float. Dôvodom je `PlotlyChart`, ktorý potrebuje hodnoty ako objekty.

Škála osi

Pri cactus plote je implementovaná i možnosť prepínať medzi škálou horizontálnej osi - osi času. Možnosti sú logaritmická mierka a lineárna mierka.

4.8.4 ScatterPlot a ScatterVisualization

Stránka s vizualizáciou pomocou scatter plotov `ScatterVisualization` tiež obsahuje tabuľku s výsledkami `ResultEntryTable`, ktorá má funkciu filtrovania, viď sekcia 4.5. Užívateľ si vyberie *hlavný riešič*, voči ktorému chce

porovnávať. Zobrazia sa grafy - scatter ploty - pre každú dvojicu riešič a hlavný riešič. Keďže v tejto vizualizácii je viacero grafov, prostredie na stiahnutie obrázkov je implementované ako vyskakovacie okno `DownloadPopUp`, kde si užívateľ zvolí, ktoré grafy chce stiahnuť v akom formáte `ImageFormats`, a stiahnu sa naraz. V prípade, že nastane nejaký problém, zobrazí sa vyskakovacie okno s chybovou hláškou vraviac čo je problém. Viď obrázok 4.4.

Škála osí

Pri scatter plotoch je implementovaná i možnosť prepínať medzi škálou oboch osí, avšak obe osi majú vždy rovnakú škálu. Možnosti sú aj tu logaritmická mierka a lineárna mierka.

Príprava dát

Trieda `ScatterVisualization` spracúva dáta z databázy do vhodného formátu pre `ScatterPlot`. Pre každý riešič rôzny od hlavného vytvorí združené výsledky `MergedResults`, viď sekciu 2.4.2 Porovnanie riešičov a obrázok 4.3. Jednotlivé typy výsledkov sú zobrazené ako samostatné časti grafu, majúc vlastné farebné označenie. Pri príprave výsledkov teda rovno rozdeľujeme dáta do samostatných častí v slovníku.

Pomocná línia

Pri scatter plote zobrazujeme i pomocnú líniu, ktorú môžeme popísať ako funkciu $f(x) = x$. Definujeme ju ako začiatočný a konečný bod. Tie určujeme pomocou extrémnych hodnôt z dát. Avšak pri logaritmickej mierke keď máme nulové hodnoty, táto línia rozhodí zobrazenie grafu. Preto v takom prípade siahneme po minimálnej hodnote rôznej od nuly.

4.9 Tabuľky

V tejto práci používame 2 druhy tabuliek. Jeden typ je pre dáta z databázy - na benchmarky a na výsledky. Druhý typ je sumarizačná tabuľka.

4.9.1 DatabaseTableTemplate

`DatabaseTableTemplate` reprezentuje tabuľku v databáze. Je to generický komponent čo sa týka typu zobrazovaných dát. Je prispôsobený na prácu s EntityFrameworkom, kde typ dát v tabuliek je reprezentovaný triedou s

properties. Táto tabuľka je závislá na tom, keďže pomocou reflection [27] získava informácie o properties danej triedy.

Keďže tabuľka by mohla byť obrovská a spomaľovala by rendrovanie, máme 2 možnosti — urobiť stránkovanú tabuľku, alebo rendrovať len úsek tabuľky ktorý momentálne vidno v okne. Vzhľadom k tomu, že scrollovanie mnohými výsledkami nie je ideálne, zvolili sme stránkovanie avšak s rendrovaním len akutálne videného úseku tabuľky. To je implementované pomocou komponentu `PagedTable`. Viď obrázok 4.5.

Properties a metódy

IgnoreColumns
: Mená stĺpcov ktoré nezobrazovať.

DbGetter
: Wrapper adekvátnej metódy triedy DatabaseApi, kvôli rôznym možným triedam dát.

SelectSet
: Má mať tabuľka komponent na výber benchmarkovej sady?

Filters
: Bude mať tabuľka filtre?

Sorting
: Má mať tabuľka možnosť triedenia?

SelectedEntries
: Getter momentálne vyfiltrovaných dát.

ChildContent
: Obsah tela komponentu.

ClearFilter
: Event callback na vyčistenie filtra.

void GetEntries()
: Vypýta si z databázy dáta podľa aktuálnych kritérií. Používa sa vo filtroch pri submitovaní.

`DatabaseTableTemplate` komponent môže obsahovať **telo**, podobne ako HTML tagy. Uloží sa do property `ChildContent`. To sa vizuálne umiestni do hlavičky tabuľky do riadku pod názvami stĺpcov. Používa sa na filtrovací riadok. Po zvážení kladov a záporov sme usúdili, že najlepší prístup bude,

aby programátor sám napísal filtrovacie polia a spojil ich s adekvátnymi premennými na držanie kritérií filtrovania.

4.9.2 Triedenie a SortCriteria

Samotný proces triedenia sa deje v databáze a je implementovaný ako súčasť triedy `DatabaseApi` — sekcia 4.4. Z našej aplikácie predáme LINQ výraz s parametrami triedenia, ktoré sa preložia na SQLite príkaz pomocou EntityFrameworku.

Triedenie funguje na základe reflection a properties tried reprezentujúcich dáta v databáze. Triediaca funkcia dostane dáta z databázy istého typu a kritériá triedenia `SortCriteria`. Tie obsahujú zoznam properties a typov triedenia, kde podľa posledného property sa bude triediť nakoniec, takže bude prvoradé.

Postupne podľa tohto zoznamu tvoríme LINQ výraz. Avšak, do metód `OrderBy()` a `ThenBy()` ako parameter treba jednoduchý getter aby sa tento výraz dal preložiť do SQLite príkazu. Taký getter musíme prv vytvoriť vo forme triedy `Expression`.

4.9.3 SortDialog

Parametre triedenia dát v tabuľke sa zobrazia po kliknutí na tlačidlo Sort vo vyskakovacom okne. Užívateľ má možnosť zvoliť prioritu triedenia stĺpcov a smer — vzostupne a zostupne. Viď obrázok 4.5.

Properties a metódy

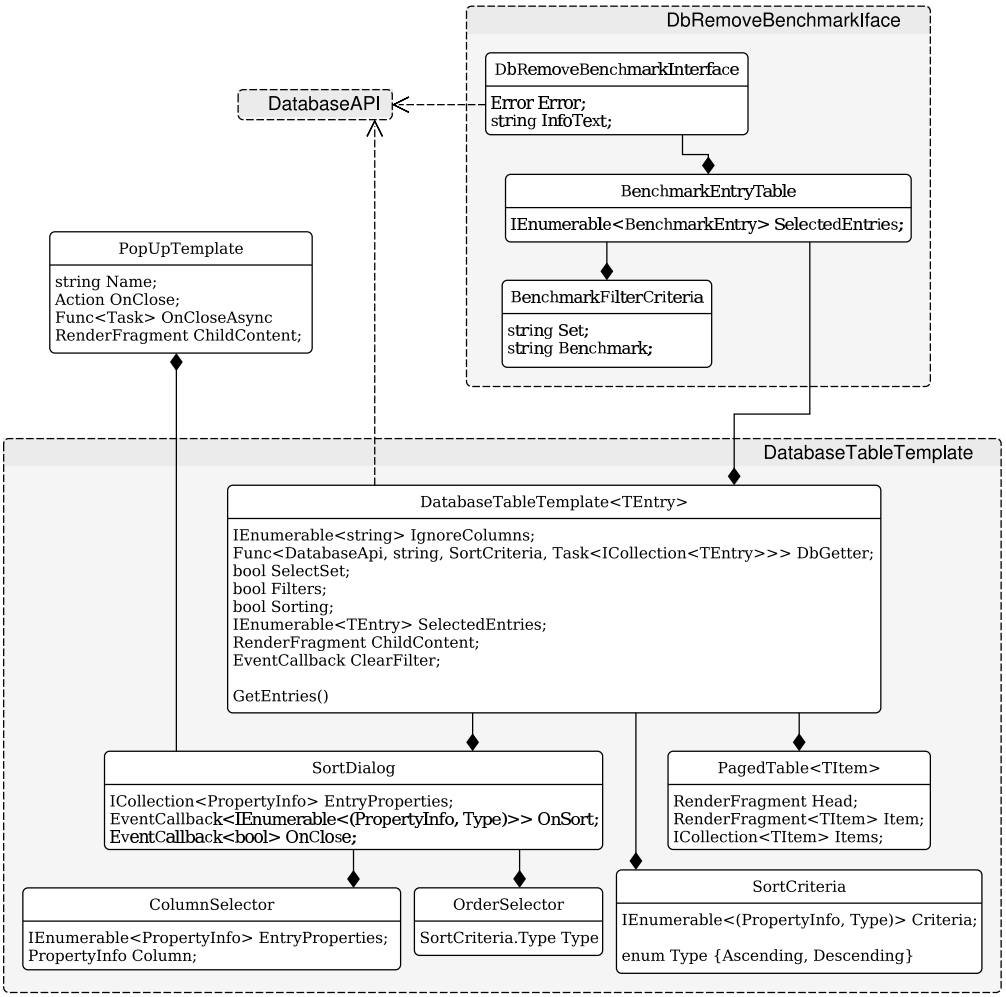
EntryProperties
: Jednotlivé stĺpce tabuľky.

OnClose
: Metóda, ktorá sa zavolá pri zavretí okna a pri zotriedení. Môže slúžiť na vyčistenie triediacich kritérií.

OnSort
: Metóda, ktorá spustí triedenie.

4.9.4 ColumnSelector a OrderSelector

Vrámci `SortDialog`-u sa používajú tieto pomocné komponenty. Sú to rozbaľovacie menu na výber ktorý stĺpec ako triediť. Oba majú property `Column`,

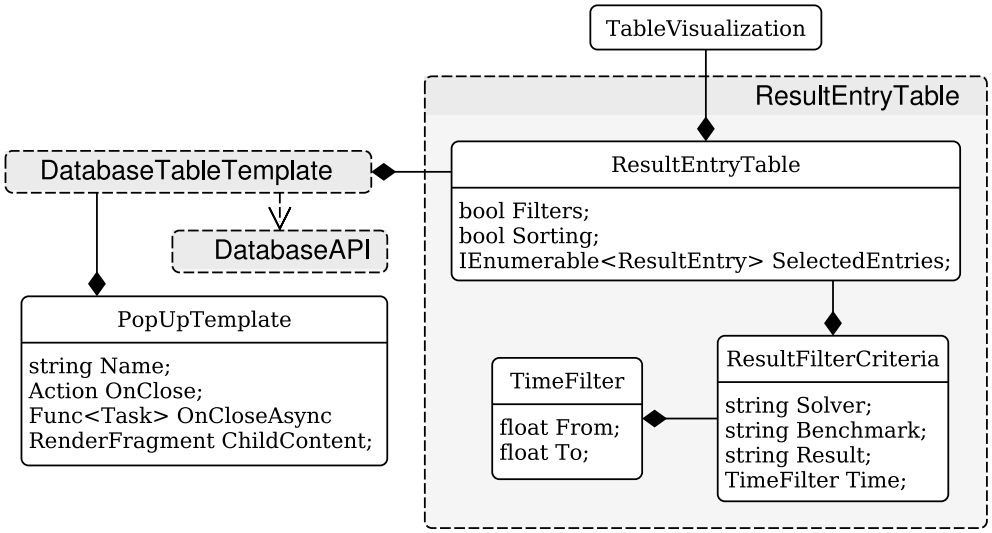


Obr. 4.5 Graf závislostí pre DatabaseTableTemplate a DbRemoveBenchmarkInterface.

ktorý obsahuje aktuálne zvolenú hodnotu. `ColumnSelector` však potrebuje ešte zoznam stĺpcov - `EntryProperties`. Viď obrázok 4.5.

4.9.5 PagedTable

Vzhľadom k veľkosti našich tabuliek sme implementovali stránkovanú tabuľku, ktorá vyrendruje len aktuálne videnú časť tabuľky pomocou komponentu `Virtualize`. Tá rozdelí dáta na zvoliteľne veľké úseky. Kvôli ich potenciálne veľkej veľkosti sme použili virtualizáciu zobrazenia. Úseky postupne zobrazuje podľa toho ako sa užívateľ posúva cez stránky tabuľky. Pre závislosti viď obrázok 4.5.



Obr. 4.6 Graf závislostí pre TableVisualization a ResultEntryTable.

4.9.6 BenchmarkEntryTable

Tabuľka s benchmarkami je špecializácia `DatabaseTableTemplate` kompomentu. Táto tabuľka neobsahuje možnosť triedenia. Viď obrázok 4.5.

4.9.7 ResultEntryTable a TableVisualization

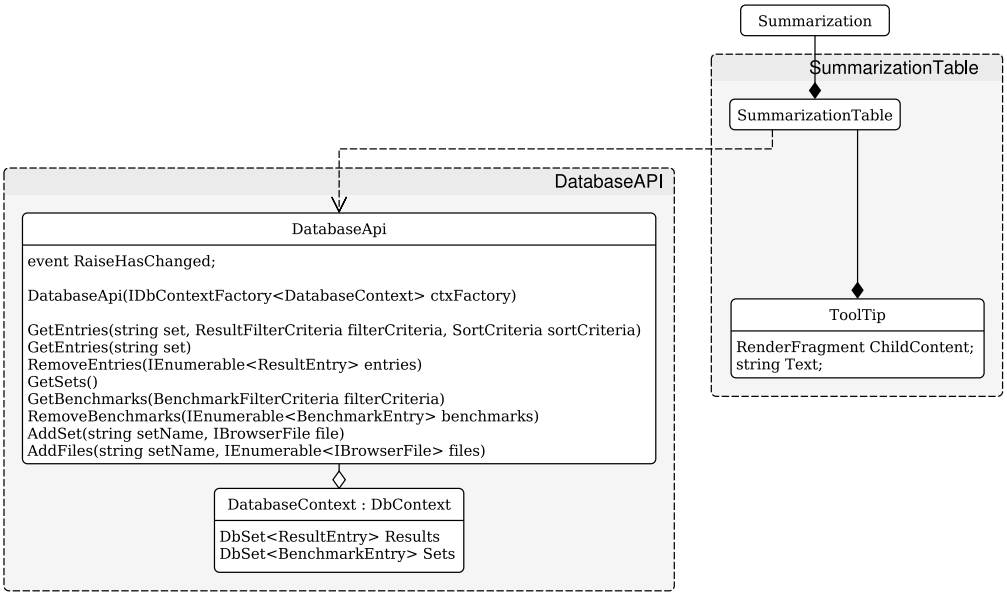
Tabuľka s výsledkami je tiež špecializácia `DatabaseTableTemplate` komponentu. V tejto tabuľke je možnosť ako filtrovania, tak i triedenia. Viď obrázok 4.6.

4.9.8 SummarizationTable a Summarization

Sumarizačná tabuľka je od `DatabaseTableTemplate` implementovaná nezávisle, viď obrázok 4.7, keďže nezobrazuje priamo dáta z databázy. Pri tejto tabuľke sú tiež dáta rozdelené podľa benchmarkovej sady ktorú si užívateľ zvolí pomocou rozbaľovacieho menu.

Pre každý benchmark prv nájdeme ktoré riešiče ho vyriešili. Nájdeme tie benchmarky, ktoré vyriešil iba jeden riešič. Následne to obrátime, a pre každý riešič zozbierame benchmarky, ktoré vyriešil iba on.

Ostatné štatistiky už vyrátame priamočiaro. Tie sú viď sekcia 2.5.2.



Obr. 4.7 Graf závislostí pre Summarization a DatabaseApi.

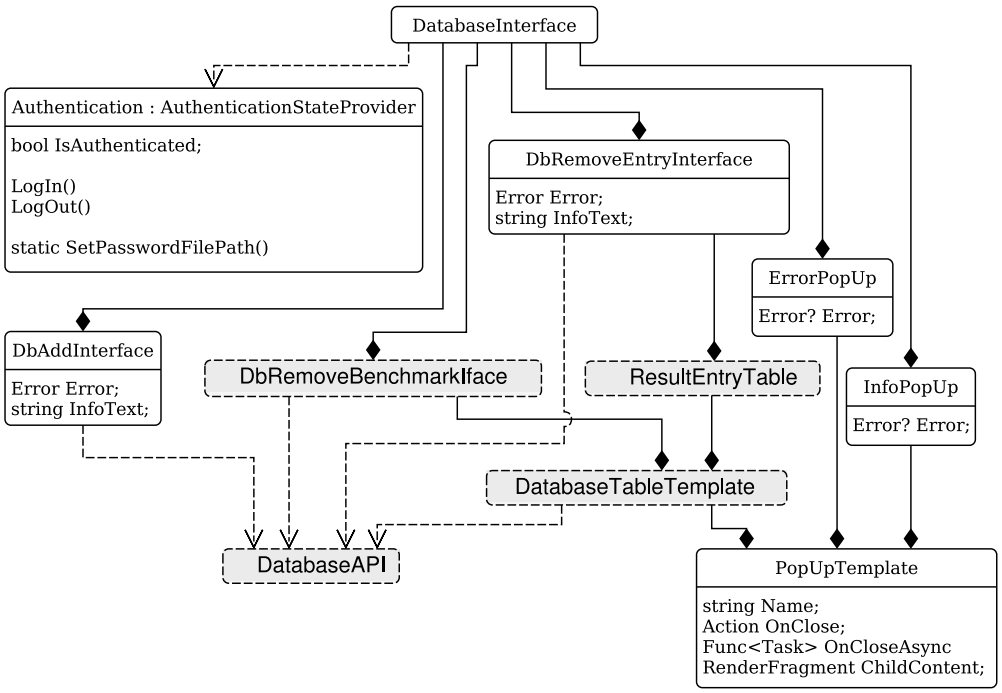
4.10 DatabaseInterface

Tieto rozhrania sprostredkúvajú komunikáciu s databázou. Prostredníctvom kariet môže užívateľ prepínať medzi pridávacím režimom, odstraňovaním benchmarkov a odstraňovaním výsledkov.

Rozhranie do databázy je chránené heslom. Pri pokuse dostať sa tam, keď užívateľ nie je autentifikovaný, bude presmerovaný na stránku s prihlasovaním `LoginInterface`. Kým sa užívateľ v databázovom rozhraní neodhlási, zostane prihlásený i po prepnutí na iné vizualizácie. Viď obrázok 4.8.

4.10.1 DbAddInterface

Pomocou rozhrania `DbAddInterface` užívateľ naraz pridáva aj výsledky aj novú sadu benchmarkov. Viď sekciu 4.2, kde je popis logiky a obrázok 4.8 pre graf závislostí. Po uploadnutí sa zobrazí informácia o prípadnom úspechu alebo neúspechu i s doplňujúcimi informáciami. Informácie sú sprostredkované cez `ErrorPopUp` a `InfoPopUp`.



Obr. 4.8 Graf závislostí pre DatabaseInterface.

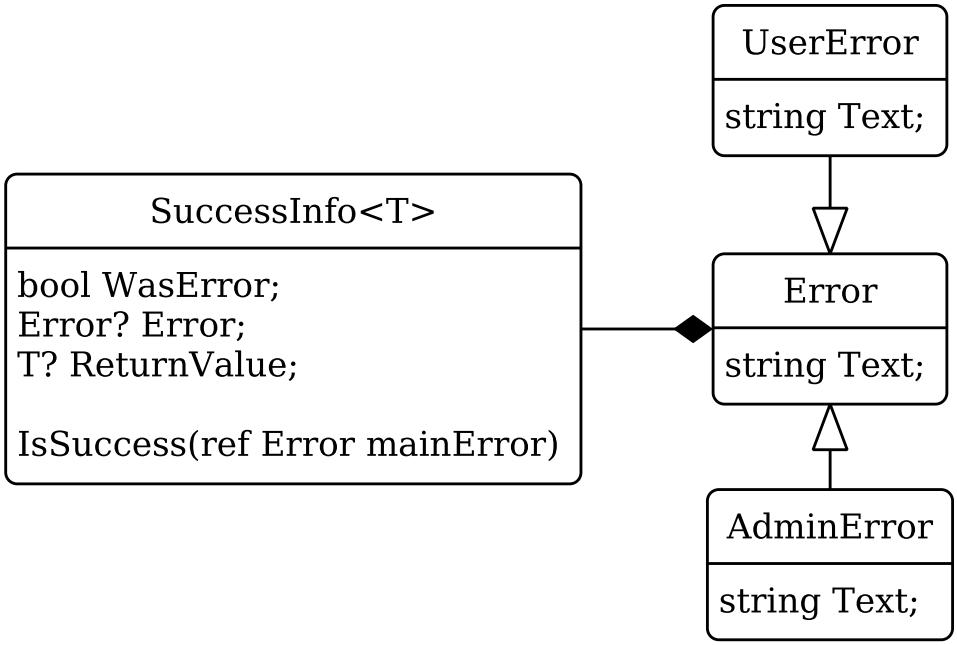
4.10.2 DbRemoveBenchmarkInterface a DbRemoveEntryInterface

Obe rozhrania obsahujú tabuľky s korešpondujúcimi dátami. Užívateľ pomocou tabuľkových filtrov zvolí ktoré dáta chce zmazať. Berú sa do úvahy všetky vyfiltrované dáta — zo všetkých stránok, nie len z práve zobrazenej. Pri odstránení benchmarkov sa odstránia i korešpondujúce výsledky riešičov. Avšak, pri odstránení výsledkov sa odstraňujú len výsledky — môže nastať nekonzistentný stav pri zobrazovaní vizualizácií.

Pre graf závislostí `DbRemoveBenchmarkInterface` viď obrázok 4.5, pre graf závislostí `DbRemoveEntryInterface` viď obrázok 4.8.

4.11 LoginInterface

Prihlasovacie rozhranie obsahuje pole na zadanie hesla. V prípade nesprávneho hesla zobrazí chybu. Po zadaní správneho hesla užívateľa autorizuje na vstup do databázového prostredia a presmeruje ho naň.



Obr. 4.9 Graf závislostí pre SuccessInfo a Error.

4.12 Ostatné

Mimo hlavných komponentov používame i malé pomocné.

4.12.1 ServerInfo

Tento komponent sprostredkúva informácie o serveri a jeho pripojení do siete. To sa využíva pri sťahovaní obrázkov.

4.12.2 SuccessInfo a Error

Pri pracovaní s našou aplikáciou môžu nastať 2 typy chýb - chyba spôsobená nesprávnym používaním užívateľom (`UserError`) a interná chyba pri spracúvaní požiadaviek (`AdminError`), viď obrázok 4.9. Užívateľskú chybu vie napraviť sám užívateľ použitím správneho postupu. Internú chybu môže napraviť len správca, prípadne čakanie a znovanačítanie.

Tieto chyby sa prenášajú od zdroja až do grafického prostredia pomocou triedy `SuccessInfo`. Tá obaľuje návratovú hodnotu a prípadnú chybu ktorá mohla nastať. Vždy obsahuje len jedno z toho.

Výpis kódu 2 Implementácia metódy SuccessInfo.IsSuccess(mainError, result).

```
// Vrati true alebo false podla toho, ci bol uspech,
// alebo nastala chyba.
public bool IsSuccess(ref Error mainError, out T? result)  {
    result = default;
    if (WasError) {
        // Ak bola chyba, prida jej text ku textu
        // hlavnej zbernej chyby.
        mainError.Text += Error!.Text;
        return false;
    }
    // Ak nebola chyba, do vystupneho argumentu ulozi
    // navratovu hodnotu funkcie, ktora vratila tento
    // SuccessInfo objekt.
    result = ReturnValue;
    return true;
}
```

Výpis kódu 3 Použitie metódy SuccessInfo.IsSuccess(mainError, result).

```
while (line is not null) {
    getLine = await GetLine(...);
    if (!getLine.IsSuccess(ref error, out line)) {
        break;
    }
}
```

Na jednoduchšie používanie tejto triedy v programe sme implementovali metódu `IsSuccess(mainError, result)`. Viď výpis 2 a výpis 3.

4.12.3 FilterCriteria

Kritériá filtrovania sú triedy korešpondujúce s triedami reprezentujúcimi dáta v databáze. Pre každé reťazcové pole obsahujú reťazcovú hodnotu reprezentujúcu regulárny výraz podľa ktorého filtrovať, a pre každé číselné pole obsahujú 2 číselné premenné reprezentujúce rozsah. Viď sekcia 4.5.

4.12.4 PopUpTemplate a InfoPopUp

Tento komponent je základnou stavebnou jednotkou vyskakovacích okien v našej aplikácii, viď obrázok 4.8. Neobsahuje žiadnu logiku.

`InfoPopUp` je špecializácia tohto komponentu. Titulok — property `Name` — má nastavené na `Information` a pri zatváraní vymaže predanú informáciu.

Properties

Name
Titulok vyskakovacieho okna — Chyba, Informácia, ...

OnClose
Synchrónna metóda, ktorá sa zavolá pri zavretí okna. Ak je nešpecifikovaná (`null`), nevykoná sa nič.

OnCloseAsync
Asynchrónna metóda, ktorá sa zavolá pri zavretí okna. Ak je nešpecifikovaná (`null`), nevykoná sa nič.

ChildContent
Obsah tela komponentu. Obsahuje samotný text vyskakovacieho okna.

4.12.5 ErrorPopUp

Toto vyskakovacie okno je špecializáciou `PopUpTemplate`, viď obrázok 4.4. Obaľuje celú aplikáciu a ako kaskádový parameter sa dostane ku každému komponentu, ktorý ho môže využiť [28].

Informáciu o chybe zobrazí pri zavolaní metódy `ProcessError(error)` v prípade, že predaný objekt chyby nie je `null`.

4.12.6 DownloadPopUp

Sťahovanie scatter plotov sa deje prostredníctvom tohto vyskakovacieho okna, viď obrázok 4.4. V ňom môže užívateľ zaškrtnúť názvy druhotných riešičov, ktorých grafy voči hlavnému riešiču chce stiahnuť. Je možnosť stiahnuť i všetky grafy pomocou špeciálneho zaškrtávacieho políčka.

Užívateľ z rozbaľovacieho menu zvolí formát v ktorom chce stiahnuť grafy. Všetky zvolené grafy sa automatizovane po jednom stiahnu.

Properties

Show
Má byť vyskakovacie okno momentálne zobrazené?

Plots
Grafy, ktoré ponúkať na stiahnutie.

4.12.7 Tooltip

Tooltip je malý komponent, ktorý pri nájdení kurzorom zobrazí doplňujúce informácie. Používame ho v sumarizačnej tabuľke, viď obrázok 4.7.

Properties

Text
: Doplnková informácia, ktorá sa zobrazí po nájdení kurzorom na komponent — zobrazený text.

ChildContent
: Obsah tela komponentu. Základná informácia, ktorá bude zobrazená stále.

Kapitola 5

Záver

Ako už bolo povedané v úvode, cieľom tejto práce bolo vizualizovať výkon SMT riešičov, aby vývojári vedeli ľahšie porovnávať prípadné zmeny vo výkone svojho riešiča. Podarilo sa nám implementovať viacero typov vizualizácií, ako grafových tak i tabuľkových, spolu s rozhraním na úpravu databázy.

Najprv sme zanalyzovali možné vizualizácie a spôsoby ich fungovania, a možné použitia užívateľom. Podľa toho sme navrhli užívateľské rozhranie stránok, použité technológie, spôsob interakcie a komunikácie medzi časťami stránok, komponentami a databázou.

Následne sa nám podarilo implementovať tieto časti vo forme Blazor komponentov, menovite scatter plot, cactus plot, tabuľkovú vizualizáciu a sumarizačnú tabuľku s vizualizáciami času behu riešičov pri jednotlivých benchmarkoch. Taktiež aj rozhranie na úpravu databázy. Ku grafovým vizualizáciám sme implementovali i možnosť stiahnuť obrázky grafov v rôznych formátoch. Celá implementácia vyžadovala preskúmanie a pochopenie fungovania frameworku Blazor ako aj SQLite a použitých knižníc.

Samotné komponenty sú vyskladané z ďalších našich komponentov, ktoré môžu byť ďalej použité. Keďže sme vo viacerých častiach použili reflection, môžu byť použité i na iné účely.

Práca na tomto projekte ani zďaleka nemusí byť hotová. Sú i ďalšie spôsoby vizualizácií a i ďalšie veličiny ktoré vizualizovať. To už však nie je v rozsahu našej práce.

5.1 Možné nadstavby

Túto aplikáciu možno ďalej rozvíjať z pohľadu funkcionality **rozšírením typov vizualizácií a veličín**, ktoré vizualizovať. Pri tomto type prác sa môže zísť napríklad funkcionalita Entity Frameworku `Migrations` [29], čo

zjednoduší ďalšie rozvíjanie databázy spolu s aplikáciou.

Ďalšia možnosť je popracovať na **zabezpečení prihlasovania** do databázového rozhrania, keďže momentálne mimo samotného hesla žiadne zabezpečenie neexistuje.

V budúcnosti môže byť potrebné implementovať prehľadovú stránku — **dashboard** — s aktuálnymi dátami na zlepšenie práce s aplikáciou.

Posledná z menovaných nadstavieb, avšak určite nie posledná možná, je pridať možnosť napojenia na **CI systém** — continuous integration — s adekvátnymi testami.

Zoznam použitej literatúry

[1] *SMT-COMP*. URL: https://smt-comp.github.io/2023/.

[2] *SMT-LIB*. URL: https://smtlib.cs.uiowa.edu/index.shtml.

[3] Martin Nyx Brain, James H. Davenport a Alberto Griggio. *Benchmarking Solvers, SAT-style*. URL: http://www.sc-square.org/CSA/workshop2-papers/RP3-FinalVersion.pdf.

[4] Dirk Beyer, Stefan Löwe a Philipp Wendler. „Reliable benchmarking: requirements and solutions“. In: *The International Journal on Software Tools for Technology Transfer* (2019). URL: https://link.springer.com/article/10.1007/s10009-017-0469-y.

[5] *C# documentation*. URL: https://learn.microsoft.com/en-us/dotnet/csharp/.

[6] *Microsoft .NET language strategy*. URL: https://learn.microsoft.com/en-us/dotnet/fundamentals/languages\#c.

[7] *LINQ overview*. URL: https://learn.microsoft.com/en-us/dotnet/standard/linq/.

[8] *SQLite*. URL: https://www.sqlite.org/index.html.

[9] *Entity Framework Core*. URL: https://learn.microsoft.com/en-us/ef/core/.

[10] *Blazor*. URL: https://dotnet.microsoft.com/en-us/apps/aspnet/web-apps/blazor.

[11] *ASP.NET Core Blazor*. URL: https://learn.microsoft.com/en-us/aspnet/core/blazor/?WT.mc_id=dotnet-35129-website&view=aspnetcore-7.0\#blazor-server.

[12] *Plotly JavaScript Open Source Graphing Library*. URL: https://plotly.com/javascript/.

[13] *Plotly.Blazor*. URL: https://github.com/LayTec-AG/Plotly.Blazor.

[14] *Magick.NET*. URL: https://github.com/dlemstra/Magick.NET.

[15] *CairoSVG*. URL: https://cairosvg.org/.

[16] *iText*. URL: https://itextpdf.com/.

[17] *Overview of ASP.NET Core authentication*. URL: https://learn.microsoft.com/en-us/aspnet/core/security/authentication/?view=aspnetcore-7.0.

[18] *Authorization Namespace*. URL: https://learn.microsoft.com/en-us/dotnet/api/microsoft.aspnetcore.components.authorization?view=aspnetcore-7.0.

[19] *AuthorizeAttribute Class*. URL: https://learn.microsoft.com/en-us/dotnet/api/microsoft.aspnetcore.authorization.authorizeattribute?view=aspnetcore-7.0.

[20] *DbContext Class*. URL: https://learn.microsoft.com/en-us/dotnet/api/microsoft.entityframeworkcore.dbcontext?view=efcore-7.0.

[21] *AddPooledDbContextFactory Method*. URL: https://learn.microsoft.com/en-us/dotnet/api/Microsoft.Extensions.DependencyInjection.EntityFrameworkServiceCollectionExtensions.AddPooledDbContextFactory?view=efcore-7.0&viewFallbackFrom=net-6.0.

[22] *Regular Expression Language - Quick Reference*. URL: https://learn.microsoft.com/en-us/dotnet/standard/base-types/regular-expression-language-quick-reference.

[23] *.NET Framework Regular Expressions - PDF*. URL: https://download.microsoft.com/download/D/2/4/D240EBF6-A9BA-4E4F-A63F-AEB6DA0B921C/Regular%20expressions%20quick%20reference.pdf.

[24] *PlotlyChart*. URL: https://github.com/LayTec-AG/Plotly.Blazor/blob/main/Plotly.Blazor/PlotlyChart.razor.

[25] *JavaScript Interoperability*. URL: https://learn.microsoft.com/en-us/aspnet/core/blazor/javascript-interoperability/call-javascript-from-dotnet?view=aspnetcore-7.0.

[26] *Overwritten parameters*. URL: https://learn.microsoft.com/en-us/aspnet/core/blazor/components/?view=aspnetcore-7.0\#overwritten-parameters.

[27] *Reflection in .NET*. URL: https://learn.microsoft.com/en-us/dotnet/framework/reflection-and-codedom/reflection.

[28] *Handle errors in ASP.NET Core Blazor apps*. URL: https://learn.microsoft.com/en-us/aspnet/core/blazor/fundamentals/handle-errors?view=aspnetcore-7.0#alternative-global-exception-handling.

[29] *Migrations*. URL: https://learn.microsoft.com/en-us/ef/core/managing-schemas/migrations/.

Zoznam obrázkov

Dodatok A

Prílohy

Digitálnymi prílohami tejto práce sú:

- `SolverVisualizer` — zdrojový kód celého programu
- `Benchmarks` — vzorové výsledky benchmarkov pre nejaké riešiče

Dodatok B

Štruktúra zdrojového kódu

```
solver-visualizer
├── SolverVisualizer ..... zdrojový kód
│   ├── BackEnd ..... triedy s funkcionalitou
│   ├── Components ..... pomocné komponenty
│   │   ├── Plot
│   │   ├── Table
│   │   │   └── Sort
│   │   └── (others)
│   ├── Pages ..... zobrazované stránky
│   │   └── Layout.cshtml ..... obsahuje JavaScript skript
│   ├── Shared
│   │   └── NavMenu ..... štruktúra navigačného menu
│   ├── wwwroot ..... root z pohľadu webovej stránky
│   ├── App.razor ..... globálne nastavenie routovania
│   ├── Program.cs ..... globálne nastavenia aplikácie
│   └── appsettings.json ..... konfiguračný súbor aplikácie
├── Benchmarks ..... vzorové výsledky benchmarkov
│   ├── (set name) ..... priečinok s názvom sady
│   │   ├── (solver name) ..... súbor s názvom riešiča
│   │   └── (solver name)
│   └── (set name)
├── Database ..... databáza
│   └── (solver-results.sqlite) ..... vytvorí sa pri nahraní dát
└── Configuration ..... konfiguračné súbory meniteľné za behu
    └── password
```

B.1 Očakávaná základná štruktúra projektu

```
solver-visualizer
├── Database ..... súbory databázy
│   └── solver-results.sqlite
├── Configuration
│   └── password ..... heslo pre databázové rozhranie
└── SolverVisualizer ..... zdrojový kód aplikácie
```

Dodatok C

Užívateľská dokumentácia

C.1 Inštalácia a spustenie

1. Na spustenie aplikácie treba mať nainštalované `.NET SDK`. Návod na inštaláciu v závislosti od operačného systému je tu[1].
2. Pripravíme súborovú štruktúru podľa sekcie B.1.
3. Nastavíme veci podľa sekcie C.2
4. V priečinku so zdrojovým kódom spustíme príkaz `dotnet run`[2], ktorý spustí aplikáciu.
5. Je možné, že nastane chyba — napríklad na Arch Linuxe — a bude treba ešte doinštalovať `ASP.NET`.

- V súbore `Configuration/password` nájdeme heslo do databázového rozhrania.

C.2 Konfigurácia

Cesty

V súbore `SolverVisualizer/appsettings.json` sú stanovené cesty ku rôznym priečinkom a súborom. Môžeme ich prepísať podľa vlastného uváženia.

[1] `https://learn.microsoft.com/en-us/dotnet/core/install/linux`
[2] `https://learn.microsoft.com/en-us/dotnet/core/tools/dotnet-run`

Heslo

Heslo sa nachádza v súbore `Configuration/password`. Prepíšeme ho na požadované a uložíme.

Nastavenie servera

V súbore `SolverVisualizer/appsettings.json` sú uložené aj nastavenia servera. Tu môžeme nastaviť URL, na ktorej bude naša aplikácia prístupná.

C.3 Používanie aplikácie

C.3.1 Grafy

Pre interpretáciu grafov viď sekcie 2.4.1 a 2.4.2.

Pod grafmi v rozbaľovacom menu zvolíme sadu, vrámci ktorej chceme riešiče porovnávať. Filtrom v tabuľke si môžeme zvoliť, ktoré výsledky chceme zobraziť. V prípade scatter plotu môžeme zvoliť, ktorý riešič bude hlavný, voči ktorému chceme porovnávať ostatné riešiče.

Grafy majú možnosť priblíženia, vzdialenia, posúvania a naťahovania:

Posunutie
: Klikneme a potiahneme kurzor po osi.

Natiahnutie
: Klikneme a potiahneme kurzor z kraja osi.

Zväčšenie výberu
: Klikneme a potiahneme kurzor v grafe.

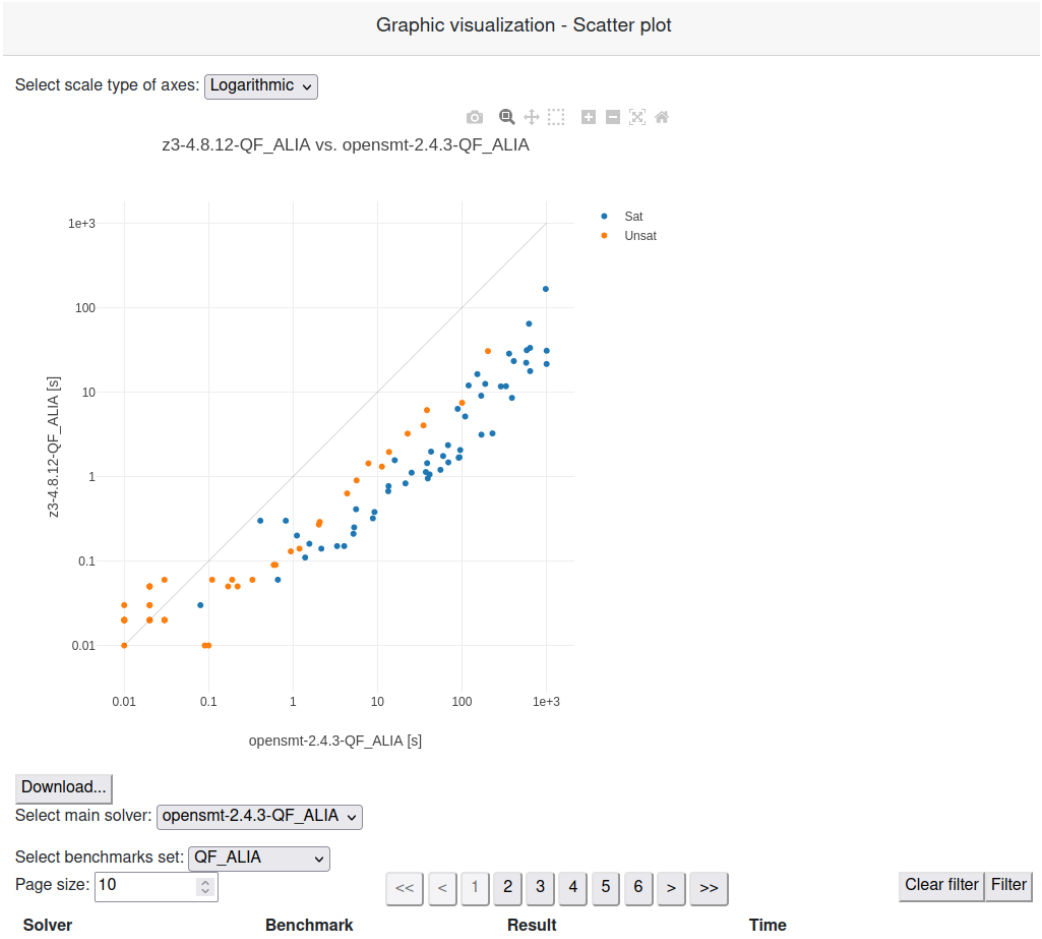
Pôvodné zobrazenie
: Urobíme dvojklik v grafe.

Posunutie
: Klikneme a potiahneme kurzor po osi.

V pravom hornom rohu sa nachádza menu na ďalšiu manipuláciu s grafom. Je tam možnosť rýchleho stiahnutia vo formáte SVG, mód posúvania grafom, približovanie, vzďaľovanie a návrat do pôvodného zobrazenia.

V hornej časti stránky sa nachádza možnosť zmeniť typ mierky osí prostredníctvom rozbaľovacieho menu.

Pod grafom sa nachádza tlačidlo `Download...`, ktoré zobrazí dialóg ohľadom sťahovania grafov, kde môžeme zvoliť formát výsledných súborov. V



Obr. C.1 Stránka s vizualizáciou pomocou scatter plotu. Vidíme možnosti zvoliť typ mierky osi, stiahnutie, voľbu sady a hlavného riešiča. Naspodku vidíme hlavičku tabuľky spolu s nastavením počtu výsledkov na stranu a navigáciu medzi stranami.

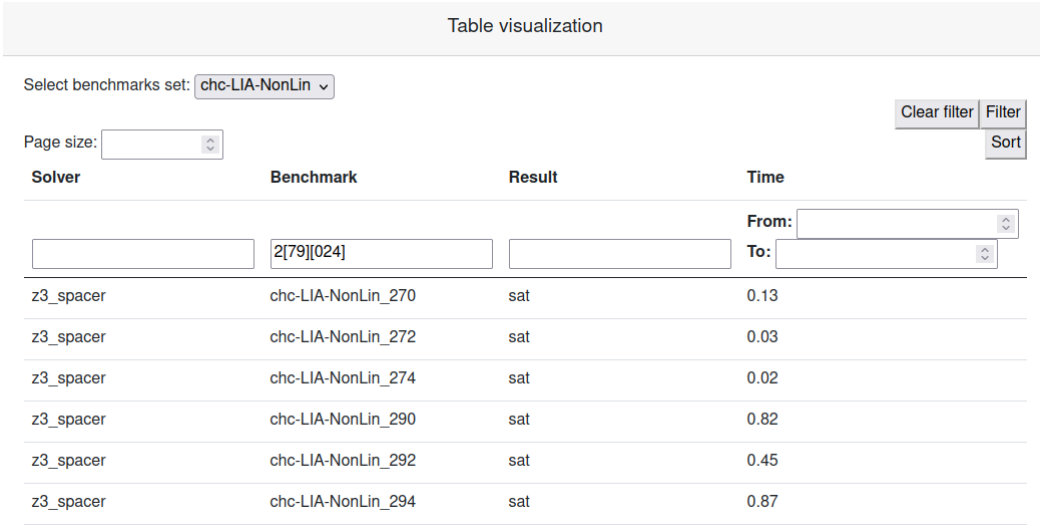
prípade scatter plotu máme možnosť naraz stiahnuť viacero grafov podľa nášho výberu. Viď obrázok C.1.

C.3.2 Tabuľkové zobrazenie

Zvolíme sadu, ktorú chceme vidieť. Pre príjemnejšie prezeranie výsledkov zadáme, koľko výsledkov chceme mať na stranu. Ak nezadáme nič, zobrazia sa všetky.

Do filtrov zadáme regulárne výrazy, podľa ktorých chceme výsledky filtrovať, viď sekcia 4.5 a obrázok C.2.

Táto tabuľka ako jediná zo zobrazených má možnosť triedenia. Po kliknutí



Obr. C.2 Stránka s vizualizáciou pomocou tabuľky. Vidíme možnosť zvoliť sadu, spôsob filtrovania s korešpondujúcim výsledkom a nestránkované zobrazenie tabuľky.

na tlačidlo `Sort` sa zobrazí vyskakovacie okno s dialógom ohľadom spôsobu triedenia. Môžeme triediť podľa viacerých kritérií. Viď obrázok C.3.

C.3.3 Sumarizácia

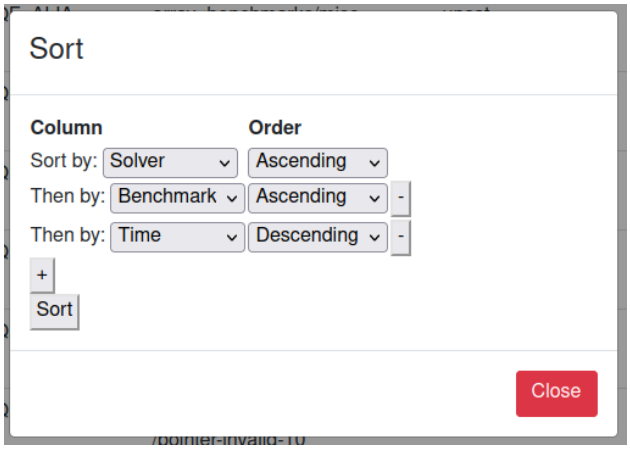
Pri zobrazení sumarizácie zvolíme sadu, vrámci ktorej chceme vidieť sumarizáciu. Pre každý riešič sa zobrazia štatistiky. Pri benchmarkoch, ktoré riešič vyriešil ako jediný, je možné kurzorom nájsť na hodnotu. Vtedy sa zobrazia názvy daných benchmarkov. Viď obrázok C.4.

C.3.4 Databázové rozhranie

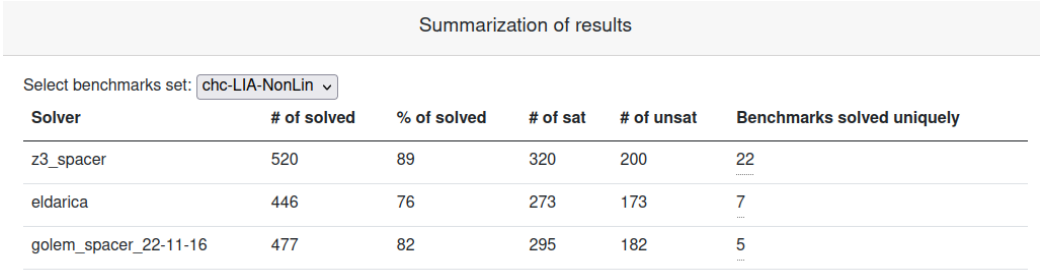
Na prístup k databázovému rozhraniu sa potrebujeme prihlásiť. Zadáme heslo, ktoré je uložené v konfiguračnom priečinku. Následne sa nám zobrazí rozhranie na úpravu databázy.

Pridávanie výsledkov na novej sade

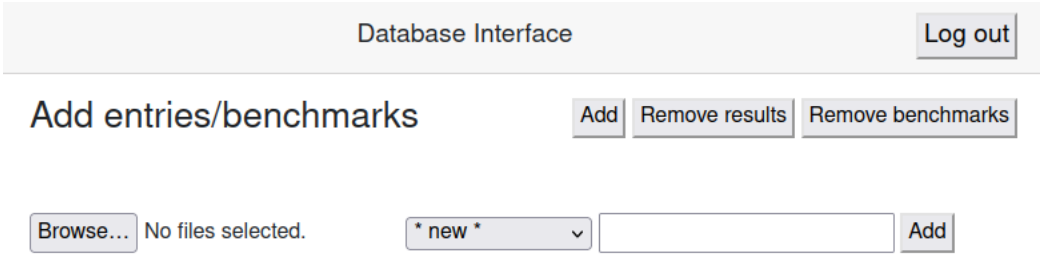
V prehľadávači súborov zvolíme výsledky riešičov na danej sade. Tieto súbory musia byť vo formáte popísanom v sekcii 2.2.1. Následne v rozbaľovacom menu zvolíme `* new *` a do textového poľa zadáme názov novej sady. Po stlačení tlačidla `Add` sa do databázy pridajú informácie ako o novej sade, tak o výsledkoch. Viď obrázok C.5.



Obr. C.3 Dialógové okno ohľadom triedenia. Vidíme možnosti pridať a odobrať stĺpce podľa ktorých triediť, taktiež voľbu smeru triedenia.



Obr. C.4 Sumarizačná tabuľka. Môžeme zvoliť sadu, ktorú chceme zosumarizovať. Po nájdení na počet jedinečne vyriešených benchmarkov sa zobrazia ich názvy.



Obr. C.5 Databázové rozhranie na pridávanie výsledkov a benchmarkových sád.

Pridávanie výsledkov na existujúcej sade

V prehľadávači súborov zvolíme výsledky riešičov na danej sade. Následne v rozbaľovacom menu zvolíme korešpondujúcu sadu. Po stlačení tlačidla `Add` sa do databázy pridajú informácie o výsledkoch.

Odstraňovanie výsledkov

Zvolíme sadu, v ktorej chceme odstraňovať výsledky. Zadaním adekvátnych regulárnych výrazov vyfiltrujeme tie výsledky, ktoré chceme odstrániť. Po stlačení tlačidla `Remove` sa odstráni informácia jedine o výsledkoch. Tým môže byť narušená jednota dát a vizualizácie nemusia zobrazovať kompletné údaje.

Odstraňovanie benchmarkov

Zadaním adekvátnych regulárnych výrazov vyfiltrujeme tie benchmarky, ktoré chceme odstrániť. Po stlačení tlačidla `Remove` sa odstráni informácia o benchmarkoch, ale aj o výsledkoch všetkých riešičov z daných benchmarkov. Tým zostane zachovaná jednota dát.