Bakalářská práce

České
vysoké
učení technické
v Praze

F3

Fakulta elektrotechnická
Katedra počítačů

Modul uživatelského rozhraní pro platformu DataGrid

Petr Tácha

Školitel: doc. Ing. Miroslav Bureš, Ph.D.
Srpen 2020

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Tácha** Jméno: **Petr** Osobní číslo: **465843**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra počítačů**

Studijní program: **Softwarové inženýrství a technologie**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Modul modul uživatelského rozhraní pro platformu DataGrid

Název bakalářské práce anglicky:

User Interface Module for the DataGrid Platform

Pokyny pro vypracování:

Navrhněte a implementujte modul uživatelského rozhraní pro platformu DataGrid, vyvíjenou ve výzkumné skupině STILL na katedře počítačů ČVUT FEL. Modul umožní rozšířit stávající uživatelské rozhraní platformy o interaktivní editor pro definici úlohy kombinatorického testování, dále umožní vypočítat několik variant testovacích scénářů (algoritmy pro výpočet scénářů jsou k dispozici v rámci platformy a není třeba je vyvíjet v rámci BP) a pomocí vhodné vizualizace uživateli poskytnout jejich srovnání. Modul následně umožní vizualizovat vybrané vlastnosti testovacích scénářů pomocí grafu. Vytvořený modul otestujte pomocí sady uživatelských testů.

Seznam doporučené literatury:

Ammann, P., & Offutt, J. (2016). Introduction to software testing. Cambridge University Press.
Kuhn, D. R., Kacker, R. N., & Lei, Y. (2013). Introduction to combinatorial testing. CRC press.
Miroslav Bures, Bestoun S. Ahmed. "On the Effectiveness of Combinatorial Interaction Testing: A Case Study." Proceedings of 2017 IEEE International Conference on Software Quality, Reliability and Security, IEEE, p. 70-76.

Jméno a pracoviště vedoucí(ho) bakalářské práce:

doc. Ing. Miroslav Bureš, Ph.D., laboratoř inteligentního testování softwaru FEL

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) bakalářské práce:

Datum zadání bakalářské práce: **24.02.2020** Termín odevzdání bakalářské práce: **14.08.2020**

Platnost zadání bakalářské práce: **19.02.2022**

doc. Ing. Miroslav Bureš, Ph.D. podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	prof. Mgr. Petr Páta, Ph.D. podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Student bere na vědomí, že je povinen vypracovat bakalářskou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v bakalářské práci.

. Datum převzetí zadání

Podpis studenta

Poděkování

Děkuji vedoucímu mé práce panu doc. Ing. Miroslavu Burešovi, Ph.D. za pomoc, kterou mi věnoval při vypracování této bakalářské práce a za doporučení odborné literatury. Také děkuji panu Ing. Václavu Rechtbergerovi za pomoc při konfiguraci projektu, na kterém je má práce založena, a za poskytnuté materiály pro rozšíření funkcí aplikace. Na závěr chci poděkovat své rodině, která mě podporovala během studia a Marii Hájkové za pravopisnou korekturu.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze, 14. 8. 2020

Abstrakt

Nedílnou součástí každého vývoje softwarové aplikace je testování. Testování je většinou časově náročné a často opomíjené. Jedna z možností jak software otestovat je pomocí kombinatorického testování a právě tato bakalářská práce se věnuje návrhu a vývoji uživatelského rozhraní webového portálu pro správu a výpočet kombinatorického testování s pomocí externí knihovny ACTS. V textu se zabývám návrhem, implementací a testováním intuitivního webového prostředí pro správu kombinatorických projektů a srovnání vygenerovaných výsledků. V práci jsou také popsány prostředky pro tvorbu moderních webových aplikací a základy teorie testování softwaru včetně vybraných algoritmů pro výpočet kombinatorických testovacích scénářů.

Klíčová slova: testování software, ACTS, testovací scénáře, kombinatorické testování, automatizované generování testovacích scénářů, webová aplikace

Školitel: doc. Ing. Miroslav Bureš, Ph.D.

Abstract

Testing is an integral part of any software application development. Testing is usually time-consuming and often overlooked. One of possible ways of testing a software is combinatorial testing and this bachelor's thesis deals with design and development of a user interface in a web portal for administration and calculation of combinatorial testing using the external ACTS library. In the text, I mainly focus on design, implementation and testing of an intuitive web environment to manage combinatorial projects and compare the results generated. The thesis is also describes the means for creating modern web applications and the basics of the software testing theory, including selected algorithms to calculate combinatorial testing scenarios.

Keywords: software testing, ACTS, testing scenarios, combinatorial testing, automated test generation scenarios, web application

Title translation: User Interface Module for the DataGrid Platform

Obsah

Obrázky

Tabulky

Kapitola 1
Úvod

Důležitou součástí každého vývoje aplikace je testování, bohužel většina projektů je testována zrychleně nebo vůbec. V mnoha případech se chyba aplikace zjistí až když se systém nasadí do provozu. Cena chyby je poté mnohonásobně vyšší, než kdyby se našla ještě před nasazením. Abychom zabránili chybovosti, je zapotřebí začít testovat aplikace již při vývoji nebo těsně před nasazením. K tomu jsou zapotřebí nemalé finance a čas, který někteří vedoucí projektů považují za zbytečný a neefektivní v porovnání s ostatními částmi projektu.

Na téma testování vzniklo mnoho studií, které se převážně zabývaly metodami, jak systém otestovat efektivně a zároveň rychle (levně). Z těchto studií se postupně vytvářely metodiky pro testování, testovací modely a přístupy pro testování nejen softwarových aplikací.

Testování hrubou silou, což je jedna z metod jak otestovat software, je velmi neefektivní a časově náročné. Jedním z mnoha přístupů, jak testování zefektivnit, je kombinatorické testování parametrů (stavů) aplikace, do kterých systém může dojít. Následné navázání těchto stavů nám dává jeden či více procesů (testovacích scénářů) k testování.

Pan Václav Rechtberger vytvořil systém pro výpočet těchto testovacích scénářů pomocí knihovny ACTS. Zaměřoval se především na serverovou část a uživatelské rozhraní vytvořil pomocí jednoduchého ovládání, které nabízela rozšířená knihovna jazyka Java. Uživatelské rozhraní bylo neintuitivní a systém počítal pouze s jediným algoritmem pro výpočet testovacích scénářů. Projekt byl zajímavý a do budoucna velmi přínosný, bohužel autor již neměl možnost projekt dál rozvíjet (zvlášť po vizuální stránce) a projekt prozatím opustil.

Mým úkolem bylo navázat na tuto stávající práci, podle potřeby upravit serverovou část systému, navrhnout a vytvořit přívětivé uživatelské rozhraní a zobrazit srovnání vygenerovaných výsledků. Nejdůležitější části systému byly správa projektu, generování testovacích scénářů v závislosti na zvoleném algoritmu a srovnání výsledků vybraných algoritmů. Jak již bylo dříve zmíněno na systému již pan Rechtberger nepracoval, tudíž jsem měl volnost ve výběru knihoven a frameworků.

Dnešní doba je především zaměřena na interakci uživatele se systémem, již pominuly doby nevhodně navržených webových stránek a velmi se dbá

na vizualizaci a intuitivním ovládání. Dalším aspektem jsou změny rozlišení obrazovky zařízení. Čím dál více lidí používá tablety, mobilní zařízení, či obrazovky s vyšším rozlišením. Aplikace není primárně určena na mobilní zařízení, ale přesto byly komponenty aplikace navrženy tak, aby si člověk, byť ne tolik intuitivně, mohl prohlédnout data i na zařízeních s malým rozlišením. Díky dnešním knihovnám a technikám webových prohlížečů je poměrně snadné tohoto cíle dosáhnout.

Text bakalářské práce obsahuje pět sekcí, v první části se zabývám teorií testování a strategiemi pro testování. V druhé části uvádím současný stav aplikace, tak jak ho vytvořil pan Rechtberger, dále je zde analýza a návrh vzhledu nového uživatelského rozhraní. Třetí část pojednává o použitých technologiích a knihovnách. Čtvrtá část se zabývá implementací systému včetně příkladů a výsledků a pátá část popisuje testování aplikace.

Kapitola 2
Testování

Většina programů je napsána lidmi, kteří se mohou dopustit chyb, ty pak mají za následek neúmyslné naimplementování defektu do systému. Tyto nechtěné prvky je potřeba co nejdříve odstranit. Problém nastává v hledání těchto defektů, kdy mnoho zanesených chyb v systému není ihned vidět a projeví se až později. A právě k tomu, abychom defekty nalezli co nejdříve je potřeba systém testovat.

Dnešní doba je především zaměřena na správné fungování systému, jelikož trh je přesycený společnostmi, které se zabývají tvorbou softwaru, je velmi pravděpodobné, že v případě nespokojenosti zákazník přejde k jinému dodavateli. Právě proto se společnosti snaží dbát na kvalitu svého produktu, aby neztratili své zákazníky.

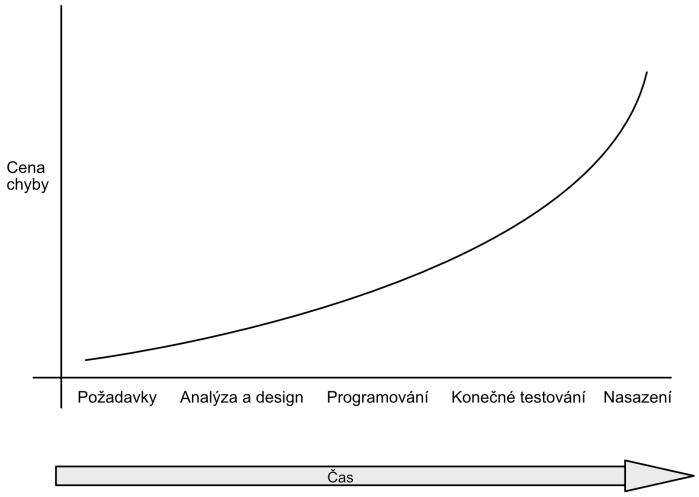
Nicméně řada firem testování stále zanedbává a přijde jim to jako ztráta finančních prostředků a času, pouze do doby, než se jim chyba vyskytne při nasazení. Společnost si náhle rychle uvědomí, jak velká cena chyby je při nasazení a jak malá cena by to byla, kdyby se defekt odchytil již na začátku.

Testování softwaru se nejen zabývá odhalením chyb v systému, ale také testuje, jestli je dodržena specifikace a požadavky na funkčnost.

Také je důležité si říci, že byť by se vytvořila sebelepší metodika pro testování, či by jsme měli téměř neomezené prostředky, nikdy nejsme schopni odchytit všechny chyby. Raději tedy mluvme o tom, že cílem testování je minimalizovat chyby v systému.

2.1 Cena chyby

Cena chyby je pojem, který určuje, jak moc cenově a časově bude stát opravení chyby pokud nastane. Klíčovými parametry jsou zde čas a fáze, ve které se projekt momentálně nachází. Cena chyby roste exponenciálně s narůstajícím časem. Proto je nebezpečné nechat testování na poslední fázi vývoje, či netestovat vůbec. Vizuálně to můžete vidět na obrázku 2.1.



Obrázek 2.1: Graf znázorňující cenu chyby s narůstajícím časem [26]

2.2 Kombinatorické testování

Otestovat software lze mnohými způsoby. Velmi často používanou metodou je kombinatorické testování. Při tomto způsobu jsou vytvořeny všechny možné kombinace stavů při průchodu programem. Zde nastává problém v počtu vygenerovaných kombinací, kdy pro systém o n parametrech, kde každý parametr by obsahoval m hodnot, je počet vygenerovaných kombinací m^n . Pro řadu systémů je nemožné otestovat všechny tyto kombinace, a proto je potřeba zvolit vhodnou strategii pro výběr kombinací, které budou zvoleny k testování. Jedna z těchto strategií je t-way testování, kde t určuje sílu pokrytí.

Princip t-way testování spočívá v tom, že každá kombinace hodnot některého z t parametrů musí být pokryta alespoň jedním testem. Tato strategie většinou zredukuje počet testovacích scénářů. Souhrnně řečeno, princip t-way testování spočívá v redukci počtu kombinačních vstupů se stále vynikající šancí odhalit chybu.

Uveďme si příklad, máme systém o 20 parametrech, kde každý parametr obsahuje 10 hodnot. Po dosazení dostáváme 10^{20} kombinací (testovacích scénářů), pokud bychom použili pairwise testování (t-way testování se sílou pokrytí 2), dostaneme se na pouhých 180 kombinací.

Zprvu se může zdát, že takové zredukování testovacích scénářů zapříčiní mnohem větší možnost, že chybu neodhalíme, nicméně studie odhalily, že většina chyb vzniklých v aplikací je vytvořená interakcí dvojic hodnot parametrů, cituji:

> **James Bach and Patrick J. Schroeder, 2004:** A pairwise test set covers 100% of the combinations of the selected values for each pair of input variables. For the two systems used in the study, random combinatorial test sets, on average, covered 94.5% and 99.6% of test data combination pairs [25].

Díky těmto studiím, se t-way testování začalo hojně používat k testování systémů.

2.2.1 IPO strategie a generování testovacích scénářů

Nejčastější variací kombinatorického testování je 2-way testování, jinak se také nazývá pairwise testování. Tato variace stačí pro otestování většiny systémů. Někdy je však potřeba systém otestovat precizněji, tj. použít vyšší sílu pokrytí. Pro t-way testování, kde je požadováno aby t bylo vyšší než 2, vznikají strategie pro výpočet scénářů. První takovou strategií je IPOG.

IPOG strategie

První strategie IPOG (In parameter order general) je zobecnění strategie IPO (In parameter order), kdy z pairwise testování se přechází do t-way testování (t>2). IPOG strategie musí zařadit do testování všechny možné kombinace hodnot parametrů, které je třeba pokrýt. Lze vypozorovat, že se zvyšováním síly pokrytí počet testovacích scénářů exponenciálně roste.

IPOG-D strategie

Jak bylo zmíněno u strategie IPOG se zvyšujícím se t exponenciálně roste i počet scénářů. K zmírnění těchto důsledků byla IPOG strategie modifikována a vznikla strategie IPOG-D. Přístup této metody se nazývá zdvojená konstrukce (nebo také D-construction), která redukuje počet t-way kombinací, které mají být zařazeny do výpočtu.

2.2.2 Shrnutí

Všechny tyto vyjmenované strategie jsou deterministické, tj. pokaždé se vytvoří stejná testovací sada pro stejnou systémovou konfiguraci.

Nicméně zatím neexistuje algoritmus, který by pro různé vstupy a parametry dával ten nejlepší výsledek (tj. nejméně testovacích scénářů), proto se dodnes vytvářejí a hledají strategie pro zjednodušení t-way testování. Prozatím si musíme vystačit s výpočtem problému pomocí různých strategií a následně porovnat výsledky, která strategie dopadla nejlépe. Knihovna ACTS obsahuje mimo jiné tyto strategie: ipog, ipof, ipof2, ipog-d.

Kapitola 3

Analýza a návrh uživatelského rozhraní DataGrid

Má bakalářská práce byla zaměřena na návrh uživatelského prostředí pro správu projektů. Návrh systému měl být intuitivní, neboť u větších projektů o několika desítek parametrů začínají jednotlivé části být velmi nepřehledné. Výhodou je, že uživatelské rozhraní lze vytvářet od základu, jelikož má práce navazuje pouze na serverovou část pro výpočet testovacích scénářů. Měl jsem možnost zvolit si, jakým způsobem a v jakém frameworku budu systém implementovat.

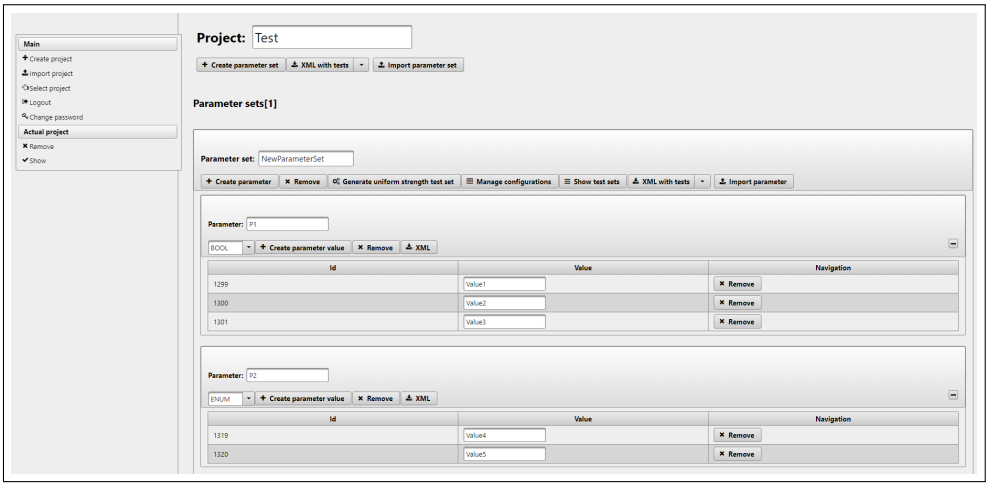
3.1 Současný stav aplikace DataGrid

Jak již bylo řečeno má práce navazovala na již vzniklý systém, tento systém se nazýval Combinatorial testing tool a byl vyvinut panem Václavem Rechtbergerem. Díky tomuto systému jsem byl schopen odvodit požadavky na systém a případná omezení systému. Serverová část využívala třídy balíčku Javax[1] a jelikož jsem s touto rozšířenou verzí jazyka Java nikdy nepracoval, byl pro mě kód velmi nejasný a nepřehledný. Pro uživatelské rozhraní se autor rozhodl použít metody Facelets.

Facelets je deklarativní jazyk, který definuje styly stránek k produkci uživatelského rozhraní. Tento přístup měl řadu výhod (rychlá kompilace, přímé napojení na serverovou logiku, vysoce rychlé vykreslování ad.). Systém byl funkční, ale uživatelské rozhraní bylo nepřehledné a nevhodně navržené viz. obrázek 3.1.

Pro přidání některého prvku do projektu se otevřelo modální okno s dodatečnými informacemi. Tento způsob přidávání prvků mi přišel velmi zdlouhavý a těžkopádný. Avšak špatný návrh uživatelského rozhraní nebyla autorova chyba, neboť ten měl za úkol pouze vytvořit zjednodušené ovládání a věnoval se více výpočetní části a struktuře systému. Struktura serverové části byla koncipována do vícevrstvé architektury, což mi velmi pomohlo při pozdějším přepisu systému, jelikož jsem ji také využíval.

[1] Rozšiřující balíček jazyka Java



Obrázek 3.1: Původní uživatelské rozhraní aplikace

3.2 Případy užití

Ze stávajícího systému a po analýze jsem vytvořil případy užití, které by aplikace měla obsahovat. Zde vyjmenuji a popíši jendnotlivé prvky, které se v systému nalézají:

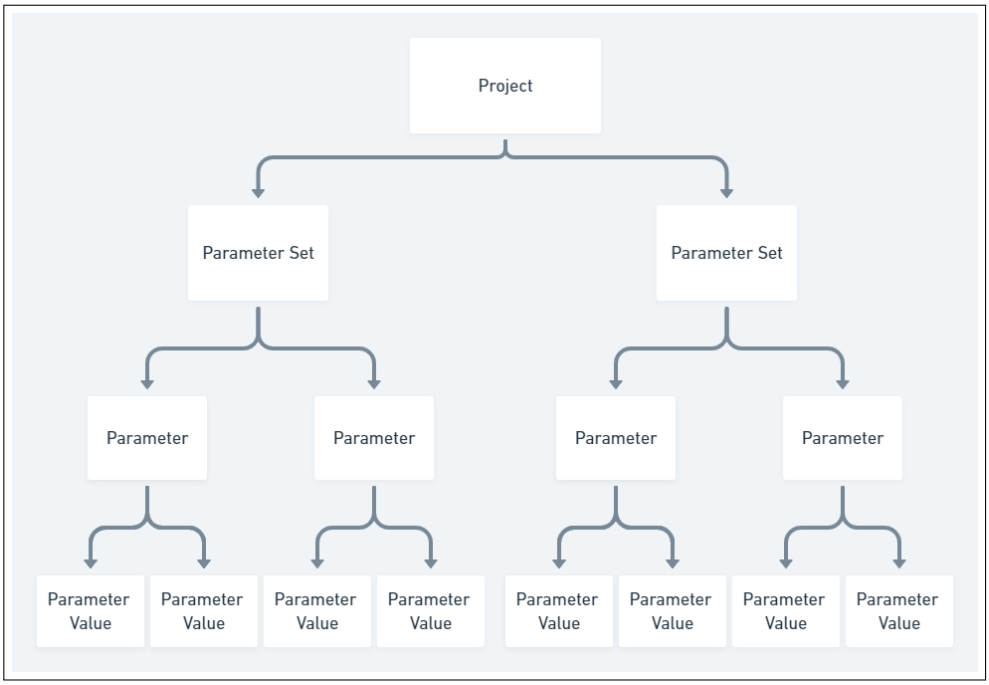
- **Uživatel**
  Funkce uživatele je zaměřena na autorizaci a autentizaci. Slouží k napojení uživatele na jeho vytvořené projekty.
- **Projekt**
  Komponenta, která se zabývá správou sady parametrů. Slouží jako specifické zaměření řešeného úkolu.
- **Sada parametrů**
  Komponenta zabývající se generováním a správou data pro vytvoření testovací sady. Skládá se z parametrů. Slouží také jako vstup pro konfiguraci a pro generování grafu se srovnáním vybraných algoritmů.
- **Parametr**
  Komponenta, která je vstupem pro výpočet testovací sady. Lze u ní nastavit typ a skládá se z hodnot parametrů.
- **Hodnota parametru**
  Závěrečná komponenta, která určuje hodnotu, kterou parametr nabývá.
- **Konfigurace**
  Slouží k bližší specifikaci generování testovací sady. Skládá se ze sad interakcí a z omezení.
- **Sada interakcí**
  Specifikuje jaké parametry mají být zahrnuty do generování testovací sady a jaká síla pokrytí mezi nimi má nastat.

- **Omezení**
  Pomocí výrokové logiky nastavuje omezení pro parametry v projektu.

- **Testovací sada**
  Komponenta, sloužící pro zobrazení dat vygenerovaných na základě sady parametrů a jejich konfigurací. Obsahuje jednu, či více testovacích sad algoritmů v závislosti na volbě generování.

- **Testovací sada algoritmu**
  Specifická vygenerovaná data v závislosti na algoritmu. Skládá se z metadat (základní informace o výsledku) a testovacích scénářů ve formátu CSV.

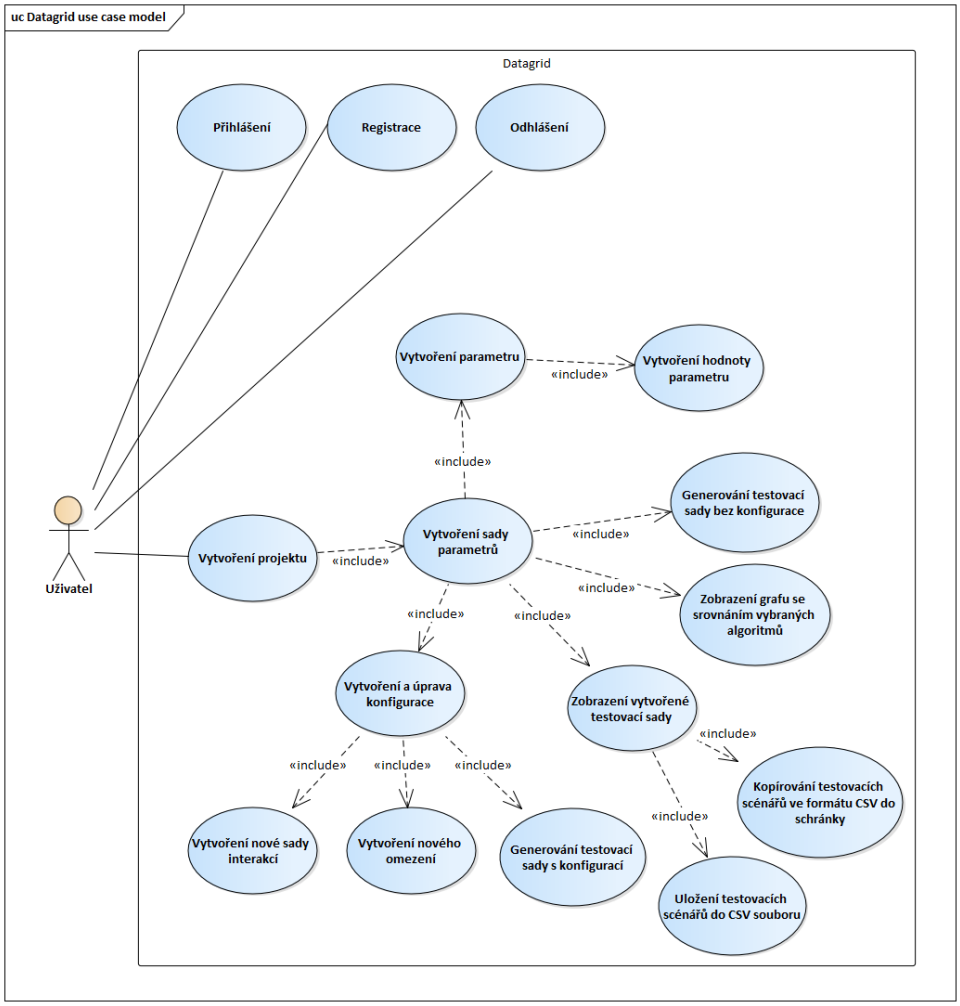
Základními případy užití jsou: Uživatel se bude moci registrovat a následně přihlásit do aplikace. Po přihlášení uživatel bude moci přejít na seznam projektů, které bude moci filtrovat podle parametrů nebo si bude moci vytvořit nový projekt. Projekt obsahuje skupiny parametrů, parametry a hodnoty parametrů. Hierarchii projektu lze vizualizovat do stromu 3.2.



Obrázek 3.2: Struktura projektu pro generování testovací sady

Veškeré parametry (jména, typy) lze editovat. Po vyplnění dat systém bude moci zobrazit graf pro srovnání vybraných algoritmů. Uživatel tak ihned uvidí, jaký typ algoritmu bude dávat nejlepší výsledky. Dále si bude moci zobrazit nastavení pro generování testovacích scénářů a také bude moci přejít do nastavení konfigurace pro přesnější specifikaci generaci testovacích scénářů. Poslední možnost nabídne uživateli zobrazit seznam již vytvořených testovacích sad včetně jejich detailů a srovnání s algoritmy. Výsledný dia-

gram případu užití na obrázku 3.3 zobrazuje potřebné interakce uživatele se systémem.



Obrázek 3.3: Diagram případu užití

3.2.1 Graf pro srovnání vybraných algoritmů

Pouhé vygenerování testovacích scénářů uživateli nenabídne příliš informací, který algoritmus dává nejlepší výsledky. Jeden z nejintuitivnějších principů zobrazení srovnání je pomocí vizuální stránky, například pomocí grafu. Pro srovnání vybraných algoritmů jsem se rozhodl použít graf, který jsem zadefinoval následovně:

osa x - obsahuje kombinace pokrytí, t-way (knihovna pracuje s možností 1 až 6).

osa y - zobrazuje počet vygenerovaných scénářů pro jednotlivé algoritmy. Osu y je potřeba zobrazovat logaritmicky, neboť pro velké projekty počet scénářů exponenciálně narůstá.

Pro srovnání byly vybrány tři nejčastěji používané algoritmy: IPOG, IPOF a IPOG-D.

3.2.2 Generování testovací sady bez konfigurace

Hlavní součástí každé sady parametrů bude generace testovacích sad. Systém pro vygenerování testovací sady potřebuje získat od uživatele tyto parametry:

- **Název testovací sady**
  Volitelná položka pro pojmenování testovací sady, pokud ji uživatel nevyplní, vygeneruje se automaticky.
- **Síla pokrytí**
  Vyjadřuje s jakým stupněm pokrytí bude testovací sada vygenerována, systém pracuje s hodnotami síly pokrytí od 1 do 6.
- **Smíšená síla**
  Určuje, jestli se testovací sada bude generovat ve stavu "mixed strength". Tato funkce umožňuje pokrýt odlišné sady parametrů s různou silou.
- **Algoritmus**
  Seznam systémem vybraných algoritmů pro generování testovací sady, včetně položky "srovnání", která vygeneruje testovací sadu se všemi vybranými algoritmy.

Po vygenerování testovací sady bude uživatel přesměrován na obrazovku detailu této sady.

3.2.3 Testovací sada

Tato komponenta slouží pouze pro zobrazení vygenerovaných výsledků na základě parametrů a jejich hodnot. Pokud došlo ke srovnání algoritmů, jsou zde zobrazeny všechny vybrané algoritmy. Bude-li zvolen pouze jeden specifický algoritmus, zobrazí se jen ten, který byl vybrán uživatelem. Každý algoritmus se bude skládat ze tří částí. Těmito částmi jsou:

- **Název algoritmu**
  Určení typu algoritmu.
- **Metadata**
  Informace o vstupních parametrech a početních výsledcích.
- **Testovací scénáře**
  Vygenerované testovací scénáře ve formátu CSV.

Pro potřeby srovnání budou algoritmy s nejlepšími výsledky názorně označeny a pro snazší získání vygenerovaných dat bude každá komponenta algoritmu disponovat funkcemi:

- Kopírování testovacích scénářů ve formátu CSV do schránky.
- Stažení testovacích scénářů ve formátu CSV do souboru.

3.3 Návrh komponent systému

Z hlediska náročnosti projektu a dnešním standardům (doporučeným postupům) bude systém rozdělen do dvou částí, a to na uživatelské rozhraní a serverovou část.

3.3.1 Serverová část

Stávající aplikace a knihovna ACTS jsou psané v jazyku Java, nicméně pro potřeby napojení na frontend a i snadnější rozšiřitelnost bylo potřeba upravit aplikaci do rest-api rozhraní. Měl jsem možnost si zvolit jakýkoliv framework, neboť stávající aplikace nebyla na žádném založena.

3.3.2 Výběr frameworku pro uživatelské rozhraní

Pro vytvoření uživatelského rozhraní bylo rozumné použít některého z moderních frameworků, jelikož použití prostého Javascriptu a HTML by bylo značně neefektivní a později by se mohl objevit problém s rozšiřitelností systému.

V dnešní době jsou na trhu desítky frameworků. Já jsem si pro srovnání zvolil tři, které jsou velmi populární, těmi jsou:

- **Angular.js**
  Framework podporovaný společností Google. Byl vydán v roce 2009 s licencí MIT[2]. Za roky se framework vyvíjel a jeho ekosystém rostl. Našel si oblibu u mnoha firem a vývojářů. Hlavním principem je vytvoření dynamického náhledu, který rozšiřuje řadu prvků běžného HTML kódu.

- **React.js**
  Největším konkurentem Angularu je framework React.js. Je to framework založený na Javascriptu a poprvé byl vydán pod licencí BSD[3] v roce 2013. Díky mnoha návodům na internetu, řadě dostupných rozšiřujících knihoven a rychlému vykreslování, se stal React oblíbený u mnoha vývojářů. React je velmi efektivní při vykreslování složitých uživatelských rozhraní, což se v případě projektu DataGrid velmi hodí. Využívá virtuálního DOMu a opakované použitelnosti komponent. Je používán například u aplikace Facebook nebo Instagram.

- **Vue.js**
  Framework Vue.js pochází od programátora jménem Evan You, který pracoval pro firmu Google, kde používal již zmíněný Angular.js. Po několika letech se rozhodl využít poznatky z frameworku Angular a vytvořil minimalistický framework pracující na návrhovém vzoru MVVM[4]. V

[2]Licence MIT je bezplatná softwarová licence, která byla vytvořena na Massachusetts Institute of Technology (MIT). https://cs.wikipedia.org/wiki/Licence_MIT

[3]"BSD (Berkeley Software Distribution) licence"je licence pro svobodný software. https://cs.wikipedia.org/wiki/BSD_licence

[4]Model – view – viewmodel (MVVM) je softwarovová architektura, který usnadňuje oddělení vývoje grafického uživatelského rozhraní (pohled) od vývoje bussines logiky nebo logiky serverové části (model). https://en.wikipedia.org/wiki/Model–view–viewmodel

roce 2014 byl vydán pod MIT licencí. Tento framework zatím nedosahuje takové popularity jako React a Angular, ale má velký potenciál do budoucnosti.

Srovnání jsem provedl pomocí tabulky:

	Obtížnost naučení	Předchozí zkušenosti	Oblíbenost na trhu	Velikost balíčku	Výkon	Přidělení paměti
Angular.js	Těžká	Žádné	Střední	4+ KB (záleží na velikosti vytvořeného svazku)	Vysoký	Vysoké
React.js	Snadná	Velmi dobré	Vysoká	116KB	Střední	Střední
Vue.js	Snadná až střední	Žádné	Vysoká	96KB	Velmi vysoký	Nízké

Obrázek 3.4: Srovnání vybraných frameworků

I když v porovnání vyhrál framework Vue.js, raději jsem si vybral framework React.js. Pro mě bylo klíčovým aspektem při výběru frameworku předchozí zkušenosti. V Reactu jsem již vytvořil pár projektů, znal jsem strukturu a jednotlivé funkce Reactu, kdežto ve frameworku Vue.js jsem nikdy nepracoval.

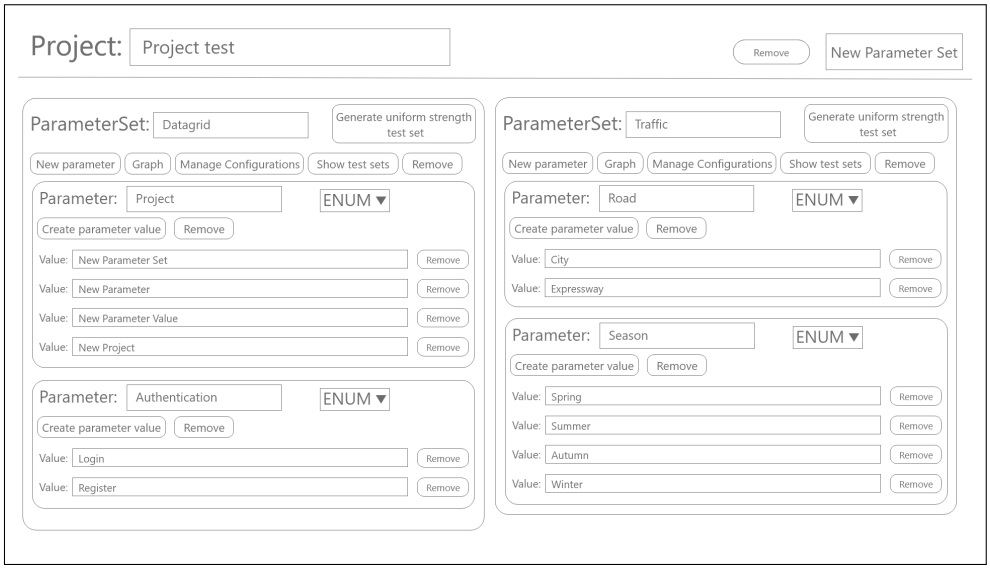
Uživatelské rozhraní jsem se rozhodl psát ve frameworku React.js.

3.4 Návrh uživatelského rozhraní

Uživatelské rozhraní můžeme rozdělit do základních sekcí. Téměř všechny důležité sekce jsou v podstatě objemné formuláře, které uživatel vyplňuje. Jelikož tyto formuláře si jsou v některých částech dost podobné je na místě použít podobné rozložení komponentů tak, aby se v nich uživatel lépe orientoval. Vhodné je využít barvy a oddělení prvků pomocí mezer. Příkladem jsou tlačítka, kde vytvoření nového prvku by mělo mít jemnou, příjemnou barvu (v systému je pro tyto účely zvolena modrá barva) a pro odstranění prvku barvu výraznou (zvolena barva červená). Důležité je také barevnou škálu nepřekombinovat, neboť by se uživatel mohl ve směsi barev začít ztrácet.

3.4.1 Obrazovka projektu

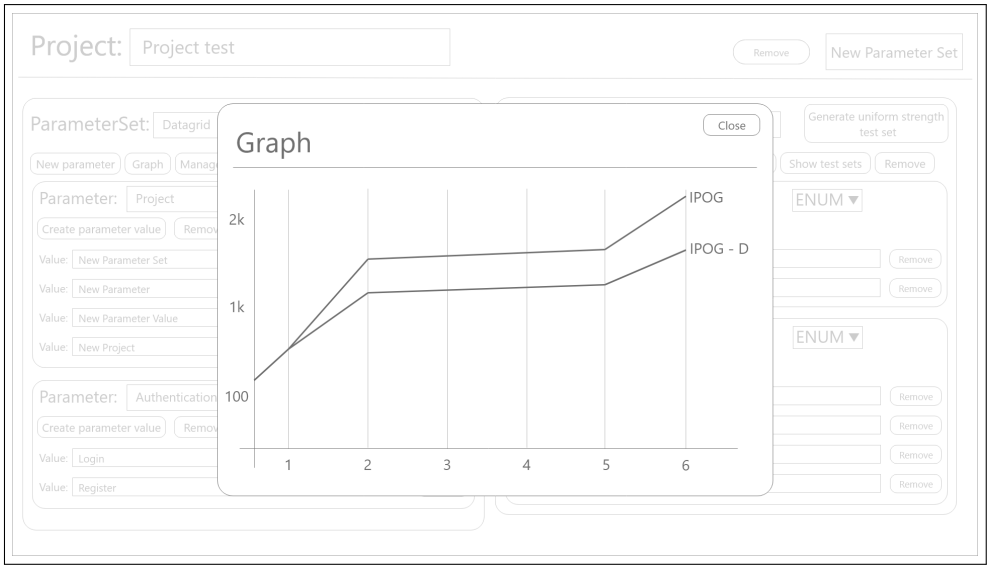
Obrazovka projektu je sekce, kde uživatel stráví nejvíce času. Tato sekce nastavuje stromovou strukturu projektu, ze kterého se budou generovat testovací scénáře, jelikož se nepočítá s použitím systému na mobilních zařízeních, lze design upravit na větší obrazovky. Návrh struktury projektu je vidět na obrázku 3.5. Pro zjednodušení systému jsou jednotlivé části rozděleny do oken, které obsahují jen nezbytné parametry, které spravují daný prvek.



Obrázek 3.5: Návrh obrazovky projektu

3.4.2 Modální obrazovka pro zobrazení grafu s vybranými algoritmy

Jedním z klíčových úkolů mé práce je srovnání vybraných algoritmů do grafu. Jelikož toto srovnání je pouze pro zobrazení jaký algoritmus dává při jaké síle nejlepší výsledky, není potřeba zde mít další funkce. Graf by měl být jednoduchý a názorný. Návrh modálního okna s grafem je znázorněn na obrázku 3.6.

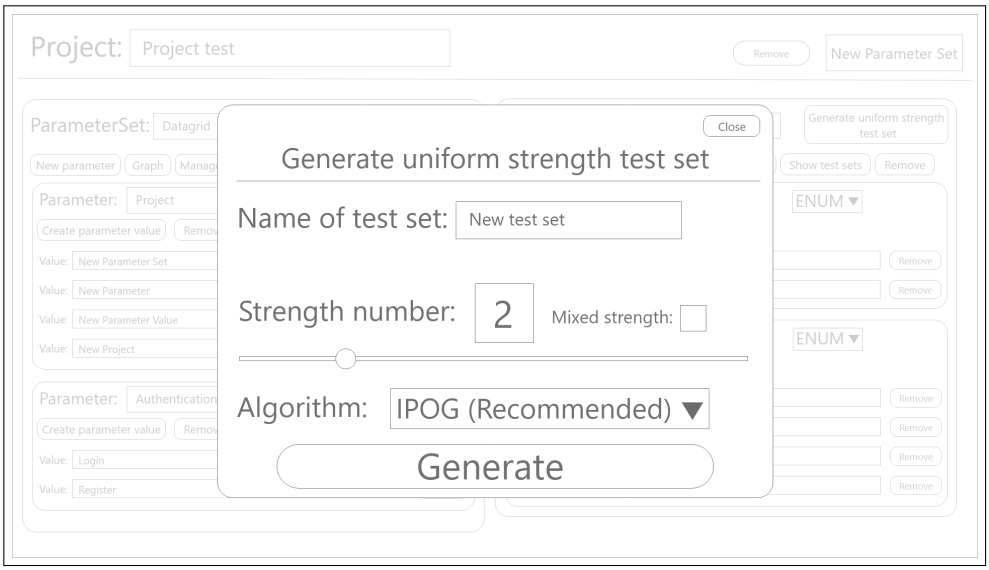


Obrázek 3.6: Návrh modálního okna s grafem pro srovnání vybraných algoritmů

3.4.3 Modální okno pro generování testovací sady

Formulář pro generování testovací sady jsem navrhnul pomocí modálního okna. Modální okna jsou v dnešní době populární, neodvádí příliš uživatele od aktuálního stavu aplikace a přesto je jejich funkčnost dostatečná pro řízení systému.

Návrh modálního okna pro generování testovací sady je na obrázku 3.7. Formulář obsahuje nezbytné parametry pro vytvoření testovací sady. Uživatel si zde zvolí jméno testovací sady, sílu pokrytí (nebo funkci mixed strength) a jaký algoritmus chce použít pro generování, či v seznamu zvolí funkci pro srovnání vybraných algoritmů.

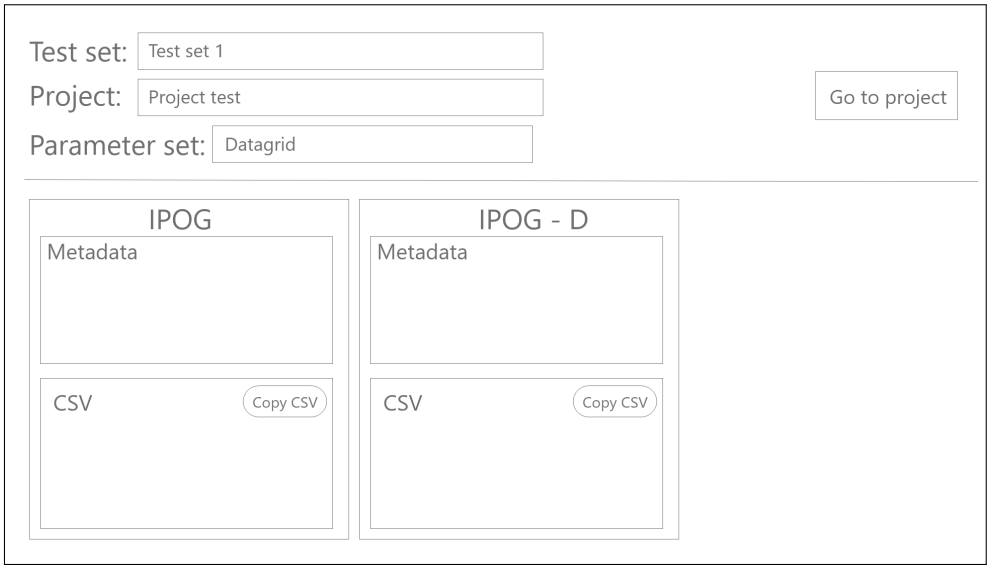


Obrázek 3.7: Návrh modálního okna pro generování testovací sady bez konfigurace

3.4.4 Obrazovka testovací sady

Poté co systém vygeneruje příslušnou sadu, je třeba tuto sadu zobrazit uživateli. Pro přehlednost jsem přidal do horní části informace o jménu testovací sady, projektu a sady parametrů včetně tlačítka pro smazání testovací sady a přechodu na projekt, ze kterého byla testovací sada vygenerována.

V druhé části jsou zobrazeny jednotlivé výsledky z vygenerovaného algoritmu nebo výsledky vybraných algoritmů. Tyto výsledky obsahují metadata (obecné informace o testovací sadě) a data testovacích scénářů ve formátu CSV. Je zde i funkce pro kopírování testovacích scénářů do schránky a možnosti uložení do souboru. Výsledný návrh obrazovky je zobrazen na obrázku 3.8.



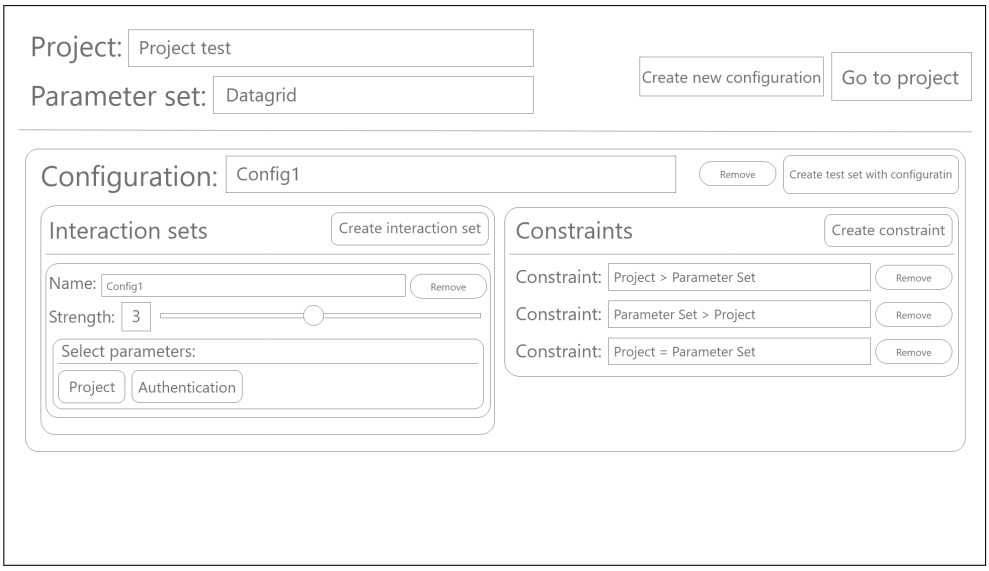
Obrázek 3.8: Návrh obrazovky testovací sady

3.4.5 Konfigurace projektu

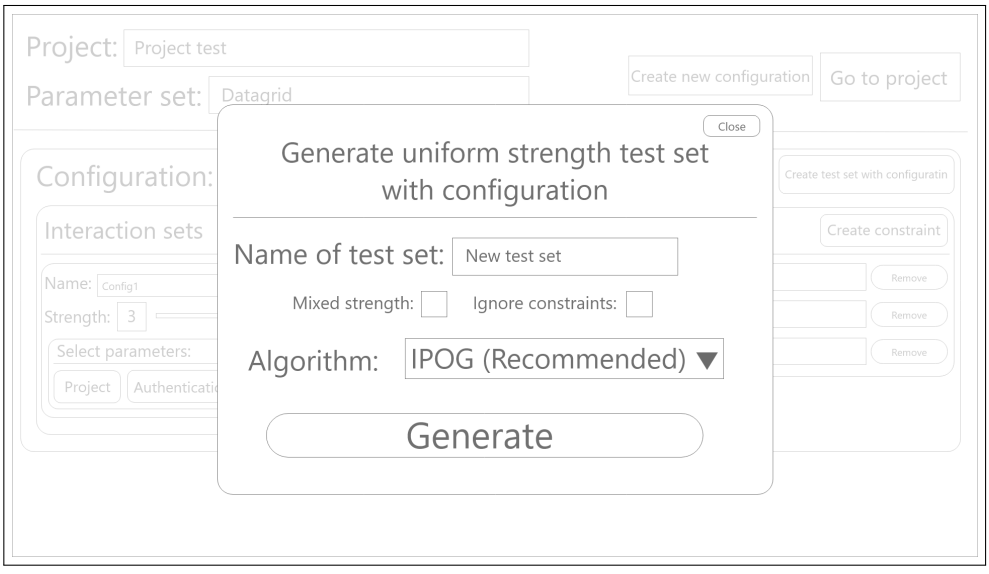
Systém ACTS také umožňuje generovat projekt s určitou konfigurací. Pro tento účel jsem raději zvolil samostatnou obrazovku, jelikož konfigurace může být velmi složitá a v modálním oknu by se mohl uživatel ztrácet. Nová konfigurace má dvě sekce, první část se nazývá interaction set (sada interakcí), kde si uživatel zvolí jméno, sílu pokrytí a jaké parametry mají být v této sadě a druhou část tvoří constraints (omezení), které slouží k omezení parametrů projektu. Návrh uživatelského rozhraní pro vytvoření omezení a sady interakcí je vidět na obrázku 3.9. Projekt může mít více konfigurací a z každé konfigurace lze vytvořit nezávislou testovací sadu. Pro vygenerování testovací sady se zobrazí modální okno, které je podobné modálnímu oknu na obrázku 3.7. Rozdíly jsou v síle pokrytí (tato síla je již zvolená u sady interakcí) a přidáním možnosti ignorovat nesprávně zadané omezení. Výsledný návrh je na obrázku 3.10.

3.4.6 Další obrazovky

Systém obsahuje další obrazovky (jako je přihlášení, registrace, hlavní stránka, přehled projektů a přehled testovacích sad). Nicméně tyto obrazovky jsou jen doprovodné a hlavní funkce systému jsou ve výše zmíněných obrazovkách.



Obrázek 3.9: Návrh obrazovky pro konfiguraci sady parametrů



Obrázek 3.10: Návrh modálního okna pro generaci testovací sady s konfigurací

Kapitola 4

Použité technologie

Většina dnešních webových systému je založena na některém frameworku, neboť vypracovávat větší systém pouze pomocí Javascriptu a HTML je neudržitelné a těžko se tento systém dále rozšiřuje. Dnešní oblíbené frameworky jsou například: VUE.js[1], React.js[2], Angular.js[3].

Z porovnání v kapitole 3 jsem se rozhodl aplikaci vyvíjet ve frameworku React.js, který staví veškerou svou funkcionalitu na znovupoužitelnosti komponent a používání stavů v dané komponentě k její přizpůsobitelnosti. React je vyvíjen firmou Facebook, tudíž je velká pravděpodobnost, že se bude dál vyvíjet a nebude rychle nahrazen jiným frameworkem.

Dále jsem použil několik knihoven, které rozšiřovali základní funkce Reactu. Těmito knihovnami jsou: Redux, React bootstrap, Chart.js, Sass, react-validation, react-bootstrap-table. Pro serverovou část jsem použil framework Spring Boot s již dříve zmíněnou knihovnou ACTS.

4.1 Framework React.js

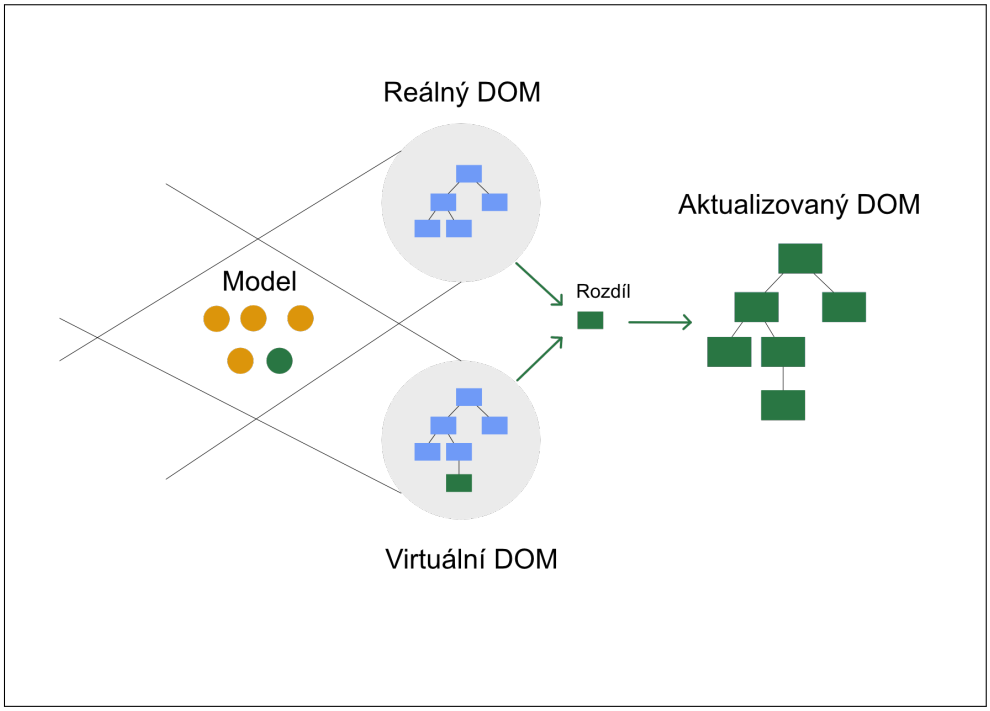
V dnešní době téměř každá webová aplikace používá jazyk Javascript k rozšíření funkcí prostého jazyka HTML. V mnoha projektech tak nastával problém s udržitelností psaného kódu a začalo se přemýšlet nad možnostmi, které by mohly tento problém vyřešit.

Jednou z těchto možností je knihovna React.js, která byla vytvořena roku 2013 firmou Facebook jako řešení náročných a velkých javascriptových aplikací. Při vytváření systému je užitečné oddělovat vizuální část od logiky aplikace. React k tomu využívá komponenty a stavy v jednotlivých komponentách.

Vykreslování v Reactu se děje efektivní cestou, kdy React srovná starou verzi DOMu[4] s novou virtuální verzí a překreslí jen ty věci, které jsou rozdílné (nové, změněné, smazané). Zjednodušený proces je zobrazen na obrázku 4.1

Každá komponenta má také svůj vlastní stav (state), který je tvořen javascriptovým objektem a lze ho měnit. Tento stav slouží pro uchování dat uvnitř komponenty a když dojde ke změně stavu v komponentě, daná

[1] https://vuejs.org/
[2] https://reactjs.org/
[3] https://angular.io/
[4] Document Object Model – objektový model dokumentu



Obrázek 4.1: Diagram principu virtuálního DOMu [27]

komponenta se znovu vykreslí. Díky tomu se oddělí logika a data jednotlivých komponent, což vede k čitelnosti a v případě potřeby ke snadné úpravě kódu aplikace.

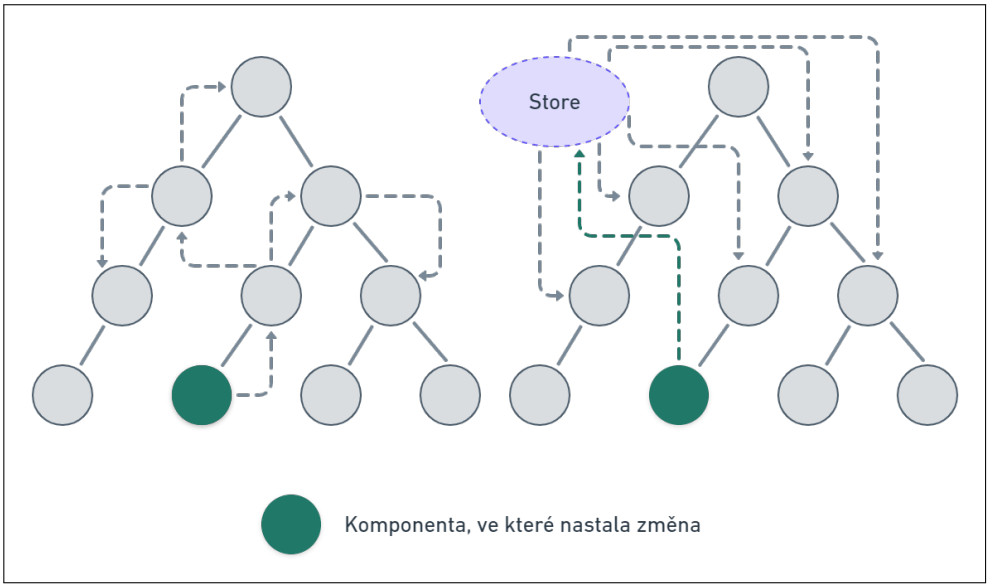
4.2 Knihovna Redux

Jak již bylo zmíněno React si ukládá data v jednotlivých komponentách, problém nastává tehdy, když potřebujeme předávat data mezi různými komponentami. Zmíněný problém je vidět na obrázku 4.2, kde vlevo vidíme způsob jakým bychom museli předávat data do jiných komponent pokud bychom používali nativní způsob Reactu, lze vidět, že pro větší systémy se stane předávání dat pomocí tohoto způsobu neudržitelné a velmi pracné.

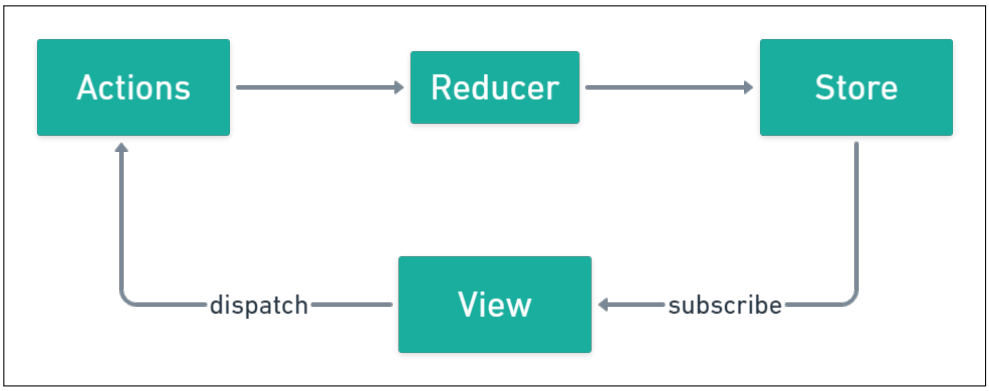
Na základě těchto problémů byla vytvořena knihovna Redux, která představuje něco jako centrální jednotný sklad, kde se uchovávají proměnné systému. Na obrázku vpravo je vidět jakou funkci Redux zastává. Komponenta uloží data do proměnné v centrálním skladu (store) a následně se přerendrují všechny komponenty napojené na tuto proměnnou. Komponenty si tak berou data z centrálního skladu a nikoliv pomocí předávacích metod. Dochází tak k udržitelnosti a přehlednosti toku dat v aplikaci.

Uvedu zde jen základní princip Redux knihovny. Způsob manipulace s Reduxem se skládá ze tří prvků a princip je vidět na diagramu 4.3.

Postup je následovný: uživatel, či některý systémový prvek vytvoří akci, která popisuje nějakou změnu v systému. Tato Akce je poté předána funkci



Obrázek 4.2: Problém s předáváním dat ve frameworku React.js [24]



Obrázek 4.3: Redux - zjednodušený princip [24]

Reducer, která se stará o to, aby byla akce přijata a její data se uložily do centrálního skladu. Centrální sklad následně předá informaci o změně stavu a všechny komponenty napojené na danou akci budou překresleny.

4.3 Spring Boot

Pro serverovou část jsem si zvolil framework Spring Boot, který vychází z frameworku Spring. Tato Open Source[5] knihovna využívá model microservices pro implementaci backendové části systému.

Microservices je název pro architekturu, která umožňuje vyvíjet jednotlivé služby odděleně, tj jsou na sobě nezávislé.

[5]tj. počítačový software s otevřeným zdrojovým kódem

Výhodou přístupu microservices se dá popsat těmito vlastnostmi:

- **Univerzálnost**
  Je možnost každou službu vyvíjet s jinou technologií, či jazykem.

- **Snazší využití programů od třetích stran**
  Pro potřeby použití jiných metod, lze snadno napojit stávající systém na programy třetích stran.

- **Jednoduší správa**
  V případě úpravy, lze měnit jen jednu část (jednu službu) a zbytek modulů ponechat beze změny.

Spring Boot je framework, který si zakládá na minimální konfiguraci pro prvotní spuštění aplikace s tím, že dodatečné konfigurace je možné přidávat časem. Výhody Spring Bootu jsou:

- **Zvýšení produktivity**
  V podobě ušetřeného času nad konfigurací systému a propojení jednotlivých částí aplikace.

- **Vyhnutí se XML dokumentům**
  Dokumenty XML jsou poměrně složité a ve většině času nepřehledné dokumenty pro správu aplikací, Spring Boot se jim snaží vyhnout.

- **Rychlá integrace s dalšími aplikacemi v ekosystému Spring Boot**
  Těmito aplikacemi jsou například: JDBC, Spring ORM, SPring Data, Spring Security a další.

Díky těmto vlastnostem je snadné vytvořit a spravovat aplikace v jazyku Java.

Hlavními důvody k výběru frameworku Spring Boot byly:

- Předešlá zkušenost s tímto frameworkem.

- Framework je poskytován zdarma.

- Snadné napojení ACTS generátoru (pomocí softwaru Maven[6]).

- Využití předchozího projektu, který byl také psán v jazyku Java.

Pro testování serverové části jsem využil Open Source program nazvaný Mockito[7].

[6] https://maven.apache.org/
[7] https://site.mockito.org/

4.3.1 Databáze PostgreSQL

Pro ukládání a správu objektů jsem využil databázi PostgreSQL. PostgreSQL je populární Open Source systém pro řízení databází. Využívá relačního modelu, což znamená, že struktura databáze je organizována do tabulek. Postgres je hojně využívaný a lze ho jednoduše napojit na Spring Boot framework. Pro potřeby mé bakalářské práce byl tento typ správy databází dostačující.

PostgresSQL jsem si vybral hlavně kvůli jednoduchosti, správě pomocí interního programu pgAdmin, ve kterým jsem již pracoval, poskytování programu zdarma a především kvůli předešlé zkušenosti.

4.4 Knihovna ACTS

Knihovna ACTS (Automated combinatorial testing software) byla vytvořena skupinou Security Components and Mechanisms. Slouží ke konfiguraci a generaci testovacích sad a scénářů. Tvůrci ji poskytují zdarma a je k dostání v souboru typu jar. Soubor jar slouží nejen jako zdroj funkcí a metod pro aplikace, které ji chtějí využívat, ale také jako samostatný program se základním uživatelským rozhraním.

Autoři přikládají ke knihovně také návod a vygenerovaný Javadoc[8], nicméně princip použití knihovny není snadný a je potřeba postupovat striktně podle přiloženého návodu, či, jak tomu bylo v mém případě, si projít jednotlivé třídy a metody a zjistit, jak (alespoň v základu) aplikace funguje.

4.5 Další použité knihovny

V aplikaci jsem použil řadu knihoven, které rozšiřovaly základní funkce Reactu a frameworku Spring Boot, upravovaly vzhled, či přidávaly komponenty, které by bylo časově velmi náročné vytvořit. Díky použití verzovacího balíčku npm[9] lze tyto knihovny jednoduše stahovat a přidávat do již existujících projektů. Zároveň se aplikace npm stará o aktualizaci těchto balíčků a řeší případné konflikty.

4.5.1 Knihovna Sass

Hlavním jazykem pro správu webové grafiky je jazyk CSS. Bohužel tento jazyk má řadu omezení a nedostatků. Jayzk Sass je metajazyk, který vylepšuje a rozšiřuje syntaxi CSS.

Sass nemá za cíl vytvořit plnohodnotný programovací jazyk pro správu designu, nýbrž má usnadnit a ulehčit práci s jazykem CSS. Hlavními výhodami jsou:

[8] softwarová pomůcka firmy Sun Microsystems™ pro automatické generování dokumentace k programu. Převzato z: https://cs.wikipedia.org/wiki/Javadoc

[9] https://www.npmjs.com/

- hnízdění (nesting)
- definice proměnných (ať se proměnné týkají velikostí, barev, či fontů)
- mixins (definice stylů, které mohou být znovu použity v jiném kontextu aplikace)
- početní operace
- podmínky

Kód pro definici stylů je díky těmto vlastnostem mnohem přehlednější a lépe se s ním pracuje.

4.5.2 Knihovna react-bootstrap-table2

Programováním a navrhováním seznamu projektů a testovacích sad jsem se nechtěl zbytečně zabývat, proto jsem si vybral knihovnu react-bootstrap-table2, která umožňuje rychle a efektivně vytvářet tabulky s daty. Kromě samotného zobrazení dat knihovna nabízí doplňkové funkce pro tabulku. Vyjmenuji zde pár funkcí:

- Snadné přidávání sloupců nebo řádků.
- Export dat z celé tabulky ve formátu CSV.
- Editace jednotlivých buněk.
- Řazení sloupců podle datumu, či abecedního pořadí.

Zvláště poslední funkci, která se zabývá seřazením položek v tabulce, jsem využil ve svém projektu, kde je vhodné, aby si uživatel mohl najít poslední upravený projekt, či si jména projektů srovnat podle abecedy.

4.5.3 Knihovna Chart.js

Tato malá Open Source knihovna je ideálním řešením pro zobrazení srovnání vybraných algoritmů v aplikaci. Splňuje všechny požadavky, které jsou definované v návrhu aplikace. Výsledné grafy dokáže vykreslit v 8 různých variantách (v závislosti na vstupních datech a nastavení), kde každý graf je responzivní. Navíc je zde stále velká možnost přizpůsobení grafu do požadované podoby a velmi snadné použití, které se skládá pouze ze tří kroků:

- Nadefinování pozice grafu, aneb kde se bude vyskytovat.
- Nastavení typu grafu a doplňkových položek ohledně os.
- Vložení dat.

Knihovna poté bez dalších potřeb graf vykreslí.

Kapitola 5

Implementace

Začátek mé práce se odvíjel z již vytvořené aplikace nazvané Combinatorial testing tool, kterou vytvořil pan Václav Rechtberger. Autor původního systému používal jazyk Java a JSX pro spuštění systému. Původní systém jsem dostal k dispozici s tím, že mohu použít jeho funkce.

Poté co jsem vytvořil základní rozhraní v Reactu a snažil se napojit na backendovou část, zjistil jsem, že to není providitelné. Autor již na projektu nepracoval, aby mi serverovou část upravil a po přezkoumání kódu jsem došel k závěru, že raději přepíšu serverovou část do mikro services za použití knihovny Spring Boot. Jelikož Spring Boot je knihovna určená pro jazyk Java a předchozí implementace byla také psána v jazyku Java, šlo většinu kódu znovu použít.

5.1 Přepis serverové části

Využil jsem poznatky z předchozích projektů a pro systém jsem použil vícevrstvou architekturu. Vícevrstvá architektura je nejčastěji rozdělena do třech vrstev. Těmito vrstvami jsou:

- Prezentační vrstva
- Aplikační vrstva (také se ji říká Business logika)
- Datová vrstva

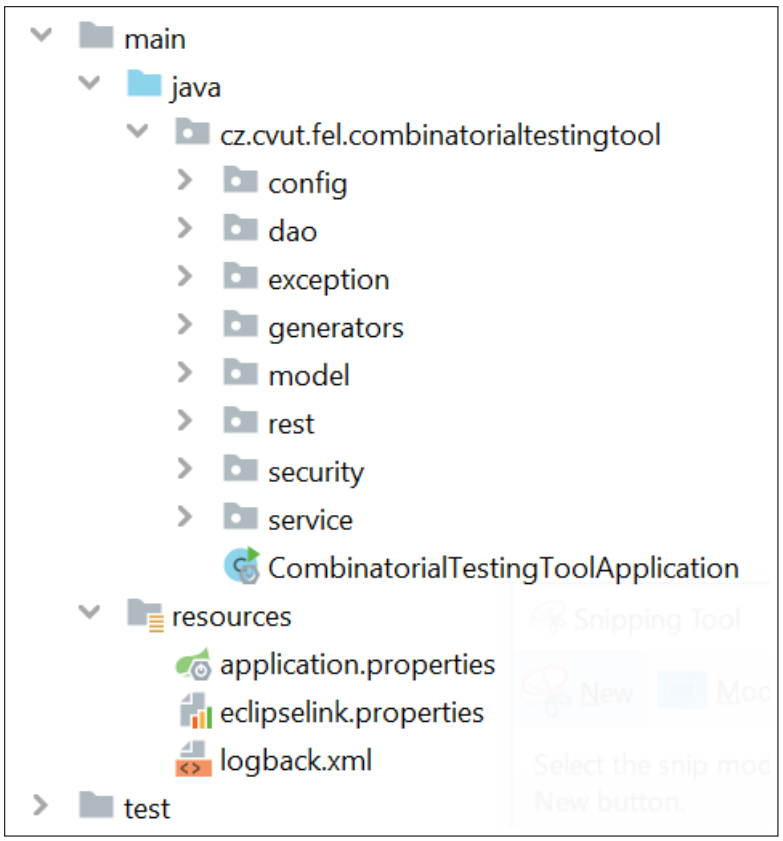
Každá vrstva zastává svou funkci. Datová vrstva pracuje s databází, vytváří dotazy a zaručuje konzistenci dat. Aplikační vrstva vytváří endpointy[1] pro prezenční vrstvu, zpracovává data a vytváří logiku aplikace. Prezenční vrstva zobrazuje data a požadavky uživateli formou některého uživatelského rozhraní.

5.1.1 Struktura aplikace

Aplikaci jsem podle pravidel rozdělil do vrstev. Inspiroval jsem se projektem, který jsme vytvořili na předmětu Enterprise architektury (zkráceně EAR) pod vedením Ing. Martina Ledvinky.

[1] Endpoint - koncový komunikační kanál

Každá vrstva se stará o jednu specifickou oblast aplikace. Na obrázku je vidět souborový systém aplikace 5.1.



Obrázek 5.1: Struktura serverové části projektu

Zde jsou popsány jednotlivé balíčky:

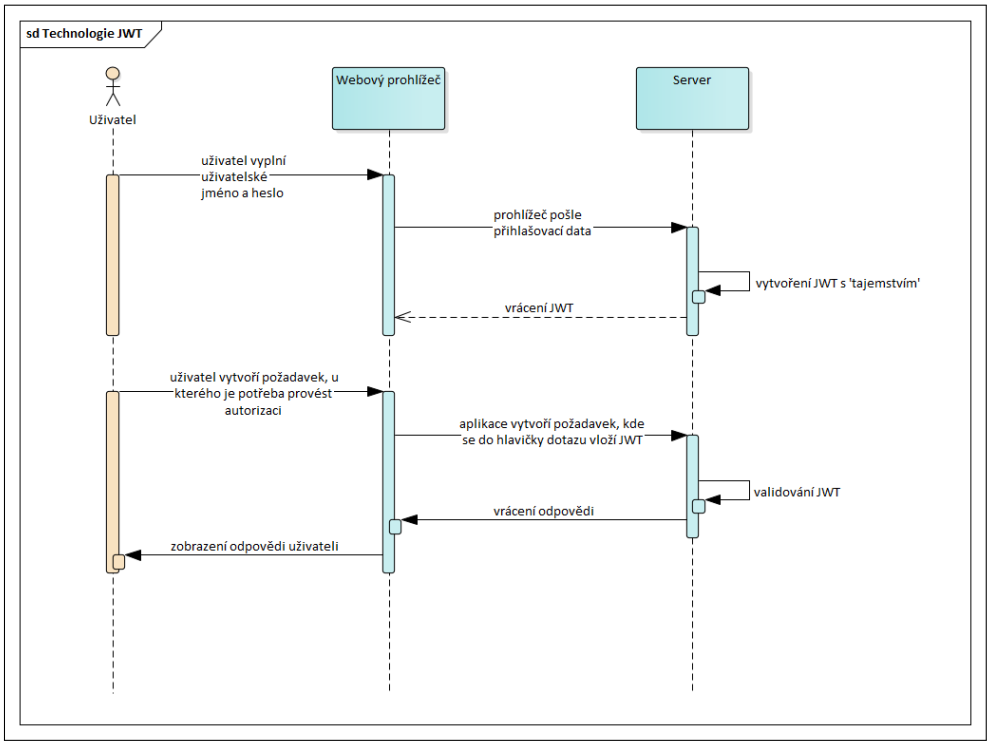
config	Tato složka obsahuje konfigurace pro knihovnu Spring. V jednotlivých třídách se nastavují parametry a metody, které se dále používají v aplikaci. Jsou to například konfigurace pro rest a perzistentní kontext.
dao	Data Access Object zkráceně DAO je oddělující prvek mezi aplikační a databázovou vrstvou. Tato vrstva pracuje s modely v databázi a vytváří na databázi sql dotazy. Obsahuje třídu EntityManager, která interaguje s perzistentním kontextem aplikace.
exception	Tato složka uchovává vyjímkové třídy. Tyto vyjímkové třídy dále slouží pro testování, či jako upozornění do konzole aplikace, že nastala chyba. Každá vyjímka obsahuje zprávu, která určuje typ a výskyt chyby.
generators	Složka generators obsahuje třídu, která se stará o generování testovacích scénářů pomocí knihovny ACTS. Také slouží pro vytvoření dat pro graf porovnávající vybrané algoritmy.
model	Zde se nacházejí třídy, které tvoří entity aplikace. Jsou také přímo mapované na databázové tabulky. Entity obsahují pouze parametry, konstruktory a getter, setter metody.
rest	Složka obsahuje kontrolery, které vytváření end pointy pro uživatelské rozhraní aplikace. Hlavní funkcí je přijetí a odeslání html požadavků.
security	Vše co se týká zabezpečení, přihlášení a registrace se vyskytuje ve složce security.
service	Service obsahuje třídy, které se starají o aplikační (bussines) stránku systému. Vytváří transakce na databázi a využívají Dao vrstvy pro správu objektů.
Combinatorial-TestingTool-Application	Hlavní třída systému spring, kde dochází ke spuštění aplikace.
resources	Obsahuje řadu souborů pro správu, tvorbu a nastavené databáze a logování.
test	Složka obsahuje testy pro aplikaci.

Tabulka 5.1: Popis struktury serverové části projektu

5.1.2 Přihlášení a registrace

Jelikož systém má uchovávat citlivá data, bylo potřeba vytvořit i proces autorizace a autentizace uživatele. Pro tento účel jsem zvolil formu JWT technologie (JSON Web Token), jehož zjednodušený sekvenční diagram je znázorněn na obrázku 5.2.

Popis průběhu je následovný: při přihlášení je poslán request na server s přihlašovacími údaji. Server následně vygeneruje JWT token a pošle ho jako odpověď uživateli. Tento token je uložen v Redux skladu a pokaždé když je posílán dotaz na server (mimo přihlášení a registrace), přikládá se do hlavičky dotazu. Díky tomu lze snadno a rychle rozpoznat uživatele.



Obrázek 5.2: Sekvenční diagram pro autorizaci a autentizaci [12]

Tato část nebyla specifikována v zadání bakalářské práce, tudíž jsem se inspiroval stránkou bezkoder.com, kde v článku *SpringBoot + React : JWTAuthenticationwithSpringSecurity* [12] jsem využil návodu pro zprovoznění této technologie. Díky tomu jsem se mohl více věnovat uživatelskému rozhraní a neztrácet čas se serverovou částí.

Uložení přihlašovacích údajů

Uživateli jsem přidal i možnost si přihlašovací údaje uložit do LocalStorage[2] prohlížeče. Jelikož je přihlašování v aplikaci napojené na Redux, bylo potřeba při příchodu na stránku vzít data z LocalStorage a uložit je do Redux uložiště.

[2]Lokální úložiště ve webovém prohlížeči

Díky této funkci se uživatel po znovu otevření aplikace nemusel přihlašovat a mohl ihned začít pracovat na projektu.

5.1.3 Napojení knihovny ACTS

Pro výpočet testovacích scénářů jsem využil část kódu, který již v systému byl. Nicméně předchozí systém pracoval pouze s jedním typem algoritmu pro výpočet testovací sady. Pro splnění zadání mé práce bylo potřeba nalézt způsob jak knihovnu nakonfigurovat tak, aby počítala s jiným typem algoritmu. Po dlouhém bádání jsem zjistil, že knihova má několik tříd, kde téměř každá třída generuje jiný typ algoritmu.

Vytvořil jsem si tedy seznam těchto tříd:

Algoritmus	Název třídy
ipog_d	BinaryBuilder
basechoice	BaseChoice
ipog, ipof, ipog_r	IpoEngine
ipof2	ForbesEngine

Tabulka 5.2: Tabulka vybraných algoritmů a k nim přiřazené třídy

Tyto třídy jsem následně napojit na stávající kód, na obrázku 5.3 je zjednodušená konfigurace knihovny ACTS.

V mém případě jsem musel tuto část upravit a přidat tam rozhodující prvek, který na základě vstupního algoritmu vytvořil příslušný engine. Výsledný kód je na obrázku 5.4 a vytvořená testovací sada díky této konfiguraci obsahovala testovací scénáře a metadata s ohledem na vstupní algoritmus.

```
// Vytvoření SUT (System Under Test)
SUT sut = new SUT ("DataGrid");

// Vytvoření nového parametru a vložením ho do SUT
Parameter parameter1 = sut.addParam("P1");

// Přidání hodnoty parametru do parametru
parameter1.addValue ("299");

// Vytvoření relace, vstupní hodnota je síla pokrytí, která
↪  nabývá hodnot 1 - 6
Relation r = new Relation (3);

// Přidání existujícího parametru do relace
r.addParam (parameter1);

// Přidáním relace do SUT
sut.addRelation (r);

// Nastavení základní síly relace
sut.addDefaultRelation (1);

// Nastavení statické třídy, která nastavuje způsob
↪  generování
TestGenProfile.instance().setRandstar(TestGenProfile.ON);

// Ignorování nevalidních omezení
TestGenProfile.instance().setIgnoreConstraints(true);
TestGenProfile.instance()
    .setConstraintMode(ConstraintMode.solver);

// Vytvoření enginu pro generování algoritmů IPOG, IPOF,
↪  IPOG_R
IpoEngine engine = new IpoEngine (sut);

// Vygenerování testovací sady
engine.build ();

// Získání testovací sady do proměnné
TestSet ts = engine.getTestSet ();
```

Obrázek 5.3: Základní konfigurace knihovny ACTS

```
        switch (algorithm) {
            case "ipof":
                IpoEngine engine2 = new IpoEngine(sut);
                engine2.buildSupportedNT(
                    TestGenProfile.Algorithm.ipof
                );
                ts = engine2.getTestSet();
                break;
            case "ipog_d":
                BinaryBuilder builderOfIpogD
                    = new BinaryBuilder(
                        sut.getParams(),
                        sut.getOutputParameters()
                    );
                ts = builderOfIpogD.getTestSet(
                    sut.getOutputParameters()
                );
                break;
            case "basechoice":
                BaseChoice bc = new BaseChoice(sut);
                ts = bc.build();
                break;
            default:
                IpoEngine engine3 = new IpoEngine(sut);
                engine3.buildSupportedNT(
                    TestGenProfile.Algorithm.ipog
                );
                ts = engine3.getTestSet();
        }
```

Obrázek 5.4: Vytvoření příslušného enginu pro vstupní algoritmus

5.2 Uživatelské rozhraní implementace

Projekt, jak již bylo zmíněno, tvoří jen několik málo obrazovek, ale každá z nich obsahuje mnoho funkcí a prvků. Pro zaručení funkčnosti Reactu a Reduxu bylo potřeba vytvořit dodatečné soubory a napojit potřebné komponenty na Redux sklad. Zároveň se musel vyřešit problém s modálními okny a upozorněním na chyby, které se v systému mohou naskytnout.

5.2.1 Autorizace a autentizace

Pro zajištění zabezpečení aplikace jsem vytvořil komponentu PrivateRoute, která se stará o kontrolu, jestli uživatel provedl přihlášení. Pokud se uživatel nepřihlásil, či mu propadl JWT token, je vždy automaticky přesměrován na přihlašovací obrazovku a zároveň je mu znemožněn přístup na ostatní obrazovky systému.

5.2.2 Modální okna

React pracuje s tezí, že každá komponenta má kořenový tag. To se děje i u celého projektu, kdy do DOMu se vloží pouze jeden tag. Obecnou zásadou je, že modální okna jsou mimo hlavní obsah stránky, je to kvůli překrytí komponent a snazší uživatelskou manipulací. Zde nastává problém s Reactem, kdy React není přizpůsobený pro modální okna a je lepší se jim, pokud je to možné, vyhnout a nepoužívat je. Pokud je přesto nutné, či velmi příhodné modální okna použít dá se to dokázat několika způsoby. Uvedu zde dva, z kterých jsem jeden použil v mé práci.

Modální okno v komponentě

Modální okno lze vložit do dané komponenty. Poté se pomocí jazyka CSS přenese modální okno do popředí. Výhodou tohoto přístupu je zachování teze Reactu, kdy je stále pouze jeden kořenový tag, práce s předáváním parametrů z a do modálního okna je snadné (okno je ve stejné komponentě a lze použít jednotný state).

Nevýhodou je nepřehlednost, kdy do specifické komponenty vkládáme prvek navíc, repetitivní vklad kódu, pokud chceme modální okno použít na jiném místě, a budoucí problémy se zobrazením a uživatelskou interakcí.

Modální okno samostatně

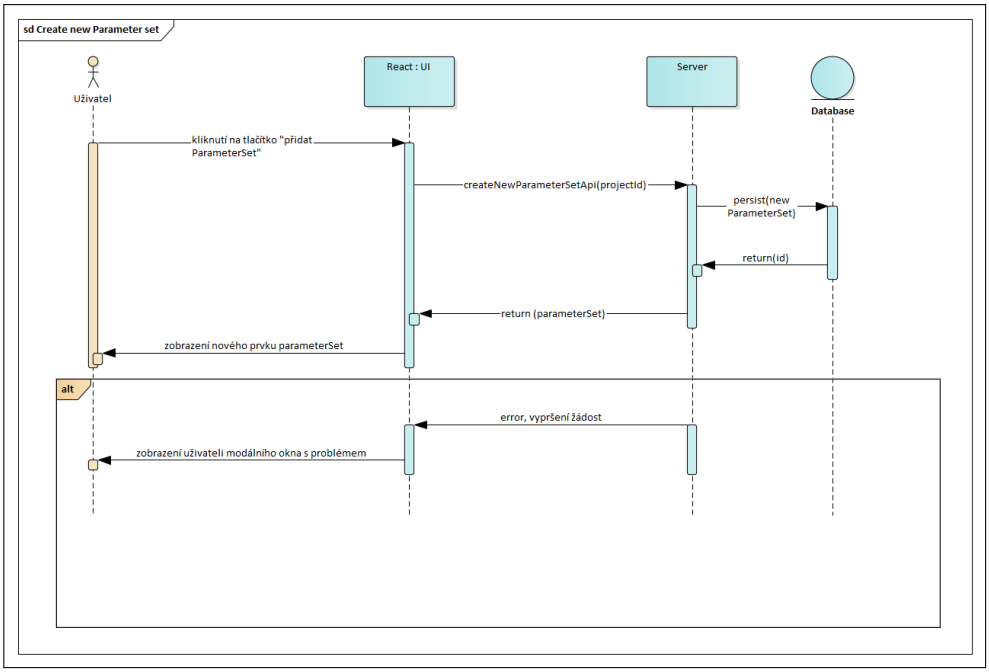
Tento přístup porušuje tezi Reactu a vytváří nový tag, který je paralelní s tagem pro hlavní obsah projektu. Díky tomu lze jednotně nastavit styl modálního okna, tak aby modální okno bylo vykresleno nad celým projektem. Modální okno pak funguje jako samostatná komponenta, která se zavolá, když je potřeba.

Nevýhodou tohoto přístupu je předávání dat, kdy modální okno a komponenta je na jiné úrovni a data se musí předávat buď pomocí props přes své rodiče nebo za pomocí knihovny Redux.

V systému DataGrid jsem zvolil tento přístup pro vytvoření modálního okna.

5.2.3 Správa projektu

Pro uživatele bylo potřeba vytvořit rychlou a jednoduchou interakci se systémem. Vytvořil jsem sadu dotazů na server, které spravují projekt. Popis průchodu jsem naznačil na sekvenčním diagramu pro vytvoření sady parametrů 5.5. Pro vytvoření projektu, parametru, hodnoty parametru a konfigurace je princip totožný.



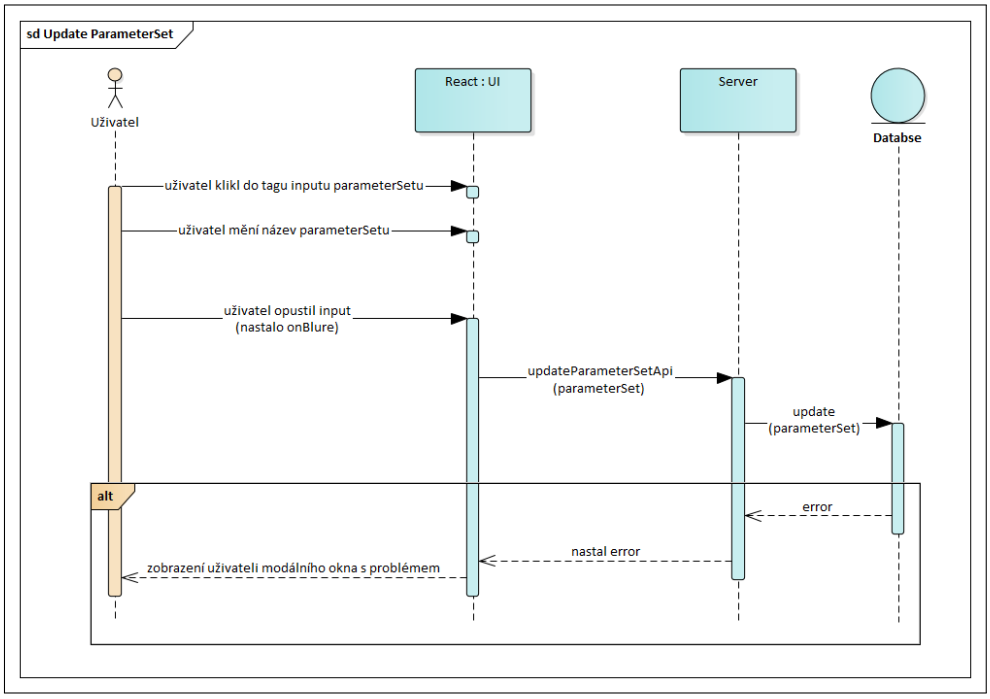
Obrázek 5.5: Sekvenční diagram pro vytvoření sady parametrů

Pro ukládání změn v projektu jsem vytvořil dotazy, které v sobě obsahují data o daném prvku (tj. id, název, popřípadě typ u prvku *parameter*), tyto dotazy pro update jsem nejdříve posílal při každé změně textu v jakémkoliv poli. Tímto způsobem se vytvořilo mnoho dotazů na server a server se tak zbytečně zatěžoval.

Chtěl jsem nalézt jiné řešení, v úvahu bylo vytvoření tlačítka, které by vzalo celý projekt a poslalo ho pomocí dotazu na server. Server by následně zaktualizoval všechny prvky. Řešení to bylo sice nejjednodušší, ale nechtěl jsem uživatele zatěžovat ukládáním a kontrolou jestli je projekt uložen nebo ne.

Vrátil jsem se k původnímu návrhu a využil jsem možnosti onBlure(), kterou React nabízí. Funkce onBlure() se vkládá do HTML tagu input spolu

s funkcí pro uložení textu. Tato funkce je zavolána právě tehdy, když uživatel opustí pole (neboli ztratí focus). To mi umožnilo poslat mnohem méně dotazů než původně a zároveň nezatěžovat uživatele ukládáním. Princip ukládání je znázorněn na diagramu 5.6.



Obrázek 5.6: Sekvenční diagram pro aktualizaci prvku ParameterSet

Na diagramu je také vidět, že pokud vše proběhne v pořádku server neposílá žádné data zpět, data zpět posílá jen v případě chyby. Tento princip funguje u všech textových vstupů v projektu a u konfigurace.

Na obrázku 5.7 je znázorněn výsledný kód tagu input pro ParameterSet.

```
<input
    type="text"
    className="parameterSet-name"
    value={parameterSet.name}
    onChange={(event) =>
        this.handleParameterSetNameChange(
            event.target.value, index
        )
    }
    onBlur={() =>
        this.updateNameOfParameterSet(
            parameterSet.id,
            parameterSet.name
        )
    }
    />
```

Obrázek 5.7: Implementace funkce onBlure

5.2.4 Graf pro srovnání vybraných algoritmů

Graf, definovaný v zadání, se měl skládat z počtu vygenerovaných scénářů na sílu pokrytí. Knihovna nabízí generování testovacích scénářů se silou pokrytí od 1 do 6, a tudíž jsem osu x rozdělil na 6 částí, kdy pro každou sílu pokrytí je vygenerována testovací sada s vybraným algoritmem.

Pro tento účel jsem musel na serveru vytvořit novou metodu, tato metoda pracuje na podobném principu jako při generování testovací sady s tím rozdílem, že má 6 cyklů (6 stupňů síly pokrytí). V každé iteraci vytvoří testovací sadu pro každý vybraný algoritmus.

Jelikož v grafu je potřeba mít pouze informaci o tom jaký algoritmus má při jaké síle celkový počet testovacích scénářů, není potřeba posílat uživateli všechny data testovacích sad. Z této skutečnosti jsem si vytvořil třídu GraphDataExporter a vnitřní třídu GraphValue (viz. obrázek 5.8). V obrázku u tříd jsem vynechal metody getters a setters z důvodu přehlednosti.

Problém bylo také získat celkový počet vygenerovaných scénářů, jelikož třída TestSet, kterou engine knihovny ACTS vytvoří, neobsahuje proměnnou s počtem vygenerovaných scénářů. Obsahuje pouze matici, kde z jejích hodnot lze zpětně získat hodnoty parametrů a vytvořit tak testovací scénáře. Nicméně velikost této matice odpovídá počtu testovacích scénářů. S tímto uvědoměním jsem získal požadovanou hodnotu pro graf.

Pro vizualizaci jsem zvolil knihovnu Chart.js. Tato knihovna je poměrně jednoduchá na použití, stačilo jenom nastavit základní parametry a vložit do

```
public class GraphDataExporter {

    private String parameterSetName;

    private List<GraphValues> graphValues
        = new ArrayList<>();

    public class GraphValues {

        private String algorithm;

        private List<Integer> values = new ArrayList<>();

    }
}
```

Obrázek 5.8: Třída pro export výsledků, které se vloží do grafu

ní data, které jsem získal ze serveru.

Základní konfigurace knihovny Chart.js je na obrázku 5.9 (příklad je převzatý z dokumentace knihovny Chart.js pro názornou ukázku [3]) a výsledné modální okno s grafem je zobrazeno na obrázku 5.10.

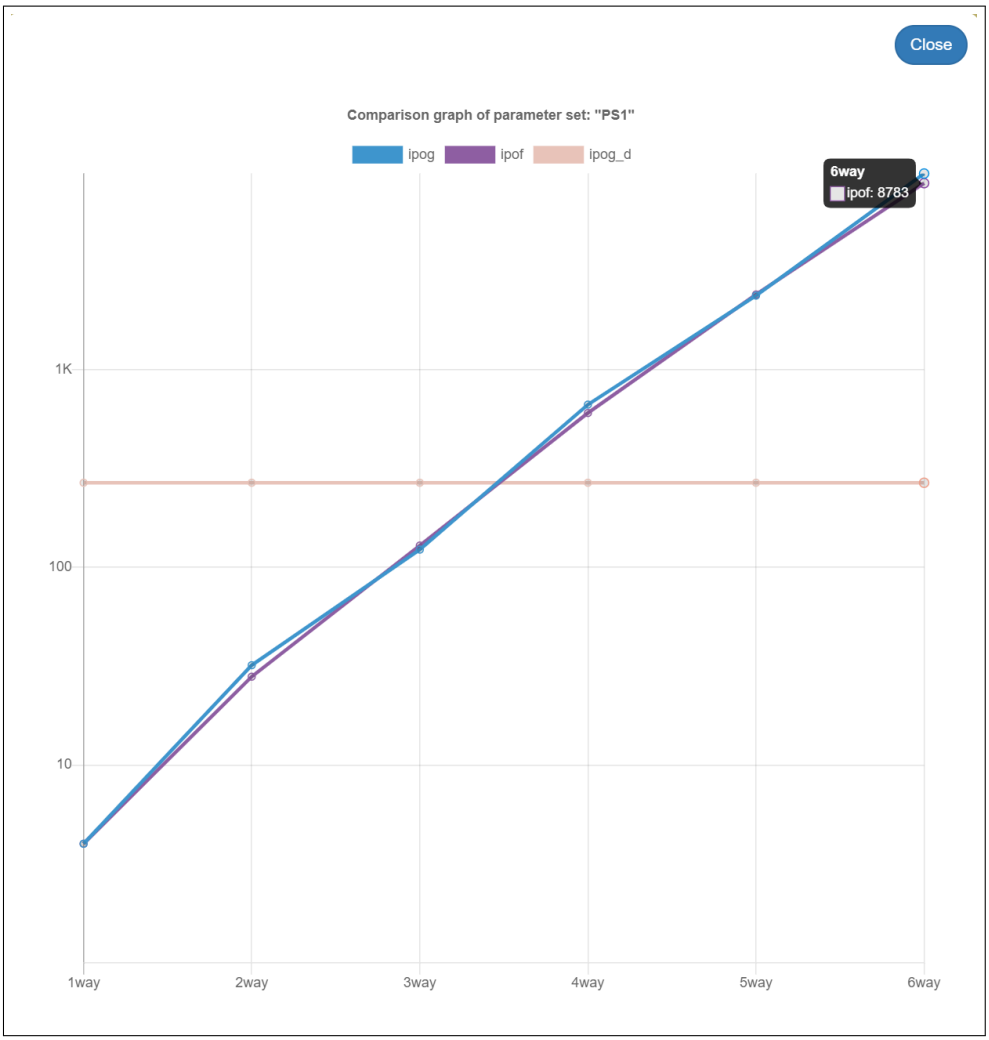
[3] Dokumentaci naleznete na této stránce https://www.chartjs.org/docs/latest/

```
// Nejdříve je potřeba si definovat tag canvas v projektu
<canvas id="graph" width="140" height="100" />

// Druhý krok je získání dat zpravidla v tomto formátu
var axisXLabel
    = [1500,1600,1700,1750,1800,1850,1900,1950,1999,2050];

// Posledním krokem je nastavení graf a vložení hodnot
 var ctx = document.getElementById("myChart");
 var myChart = new Chart( ctx, {
    type: 'line',
    data: {
        labels: axisXLabel,
        datasets: [
            {
                data: myData
            }
        ]
    }
});
```

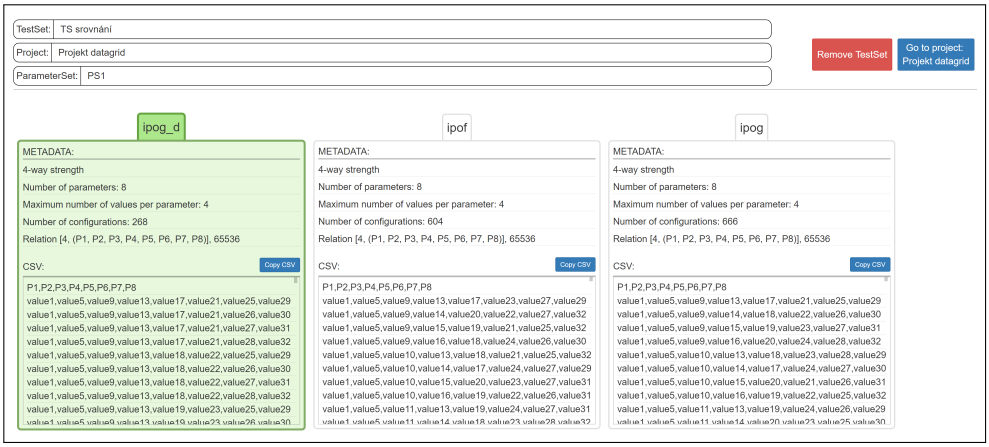
Obrázek 5.9: Konfigurace grafu knihovny Chart.js



Obrázek 5.10: Vytvořený graf pro srovnání algoritmů

5.2.5 Zobrazení testovací sady

Po vygenerování testovací sady bylo potřeba přijatá data zobrazit uživateli a zvýraznit, který algoritmus dal nejlepší výsledky. Pro jednoduchost a názornost jsem zbarvil algoritmy s nejlepšími výsledky zelenou barvou. To dá uživateli jasně najevo, že se jedná o ten nejlepší. Pokud je více algoritmů, které mají stejný počet nejméně vygenerovaných scénářů, zbarví se do zelena všechny nejlepší algoritmy. Toto se stává především u malých projektů, kde počet parametrů je menší než 2 a síla pokrytí je menší než 3. Výsledný design lze vidět na obrázku 5.11.



Obrázek 5.11: Obrazovka testovací sady se srovnáním všech algoritmů, se kterými DataGrid počítá

Funkce Copy

Většina uživatelů systému chce dále pracovat s vygenerovanými výsledky, respektive s testovacími scénáři. Velká obliba je práce s daty ve formátu CSV, tudíž bylo potřeba vytvořit funkci pro kopírování těchto dat do schánky. Našel jsem pár knihoven, které zaručovali tuto funkci, ale již nepracovali s další úpravou dat, kteoru jsem potřeboval. Jednou z těchto knihoven je například *react−copy−to−clipboard*[4]. Jelikož mi tato knihovna nevyhovovala, použil jsem čistý Javascript, ve kterém existuje metoda *document.execCommand*(”*copy*”), která vezme právě označený text a zkopíruje ho do schránky. Tato funkce je od roku 2015 podporována téměř ve všech webových prohlížečích.

Uživatele jsem nechtěl zatěžovat označováním textu a jelikož jinou metodu jsem nenašel musel jsem problém vyřešit tak, že jsem si vytvořil prvek textarea, do něj jsem vložil CSV data a následně na tento prvek jsem zavolal metodu *.select*(), která označí text, který je uvnitř. Poté co je text označený stačí zavolat výše zmíněný příkaz *document.execCommand*(”*copy*”) a uživateli oznámit, že jeho schránka obsahuje jeho vybraná data formou změny popisu tlačítka. Přidal jsem tam i časovou prodlevu, kdy po uplynutí krátké doby se

[4] https://www.npmjs.com/package/react-copy-to-clipboard

popis tlačítka vrátí zpět do původní podoby. Výsledná funkce je zobrazena na obrázku 5.12.

```
copyToClipboard = (csvData, index) => {
    let csvUpravenaData = csvData
      .map((row) => {
        return row + "\n";
      })
      .join("");
    var copyArea = document.createElement("textarea");
    document.body.appendChild(copyArea);
    copyArea.value = csvUpravenaData;
    copyArea.select();
    document.execCommand("copy");
    document.body.removeChild(copyArea);
    let currentState = { ...this.state };
    currentState.copySuccess = index;
    this.setState(currentState);

    setTimeout(
      function () {
        this.setState({ copySuccess: "" });
      }.bind(this),
      2000
    );
  };
}
```

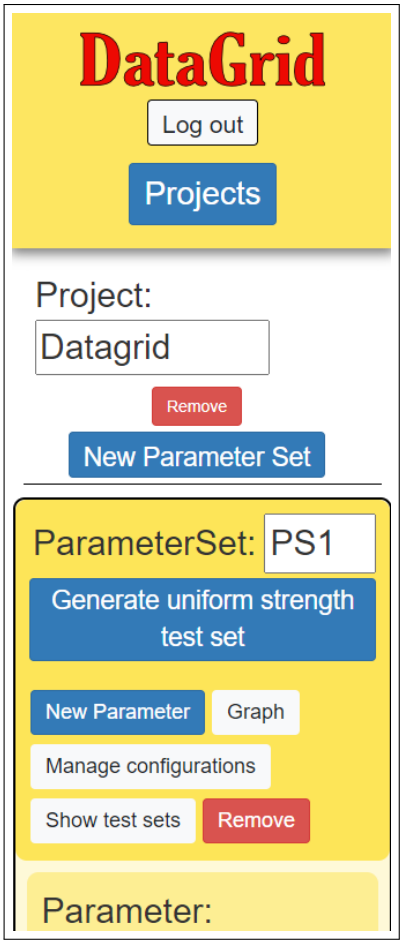
Obrázek 5.12: Kód funkce pro kopírování do schránky

5.2.6 CSS a přizpůsobení pro zařízení o různých rozlišeních

Jak bylo zmíněné ve specifikaci, systém se bude primárně používat na stolních zařízeních, maximálně tabletech. Nepředpokládá se, že by uživatelé systém používali na mobilních zařízeních. Díky tomuto zjištění jsem se více zaměřoval na stylizaci pro zařízení s větším rozlišením.

Nicméně díky dnešním technologiím, které jazyk CSS a webové přohlížeče nabízejí, jsem byl schopen poměrně jednoduše upravit uživatelské rozhraní tak, aby se i při nízkém rozlišení dalo používat.

Výsledný design při rozlišení 330x850px je na obrázku 5.13.



Obrázek 5.13: Rozložení prvků v uživatelském rozhraní při nízkém rozlišení

Kapitola 6
Testování

V této kapitole uvedu způsoby testování, které jsem použil napříč aplikací. Testování jsem rozdělil do dvou částí, a to testování serverové části a testování uživatelského rozhraní. U každé bylo potřeba otestovat jak základní funkce tak i vygenerovaná data a průběh předání dat mezi serverem a uživatelským rozhraním.

6.1 Testování serverové části

Jelikož jsem serverovou část přepisoval, bylo na místě vytvářet i testy pro kontrolu funkcí. Výhoda použitých knihoven byla ta, že při každém pokusu o sestavení projektu se spustily všechny napsané testy. Pokud některý z testů selže systém na něj upozorní uživatele, to zaručuje větší šanci odchytit chybu při vývoji aplikace, než po jejím nasazení, kde je riziko chyby daleko závažnější.

Testování ve frameworku Spring Boot není nijak náročné, spíše je pracné. Vytvořil jsem sady integračních testů, které pracují s databází h2[1]. Výhodou databáze h2 je způsob uložení dat, kdy se struktura databáze vytvoří pouze v paměti a po skončení testování se smaže. To nám umožňuje pokaždé začínat s prázdnou databází.

6.1.1 Jednotkové testy

Jednotkové testy také nazývané jako Unit tests, jsou testy určené pro otestování jednotlivých tříd a metod. Hojně se zde využívá funkce mockování, což nám umožní izolovat objekt od ostatních částí aplikace. Takový objekt se nazývá mock a chová se stejně jako pravý objekt v systému. Jinak řečeno mockování je vytváření objektů, které simulují chování reálných objektů v aplikaci.

V mém případě jsem tyto testy použil pro testování třídy uživatele a pro testování knihovny ACTS. Na obrázku 6.1 je uveden příklad testu s mockováním, test sloužil k ověření funkčnosti nalézt uživatele pomocí jména.

[1] https://www.h2database.com/html/main.html

```
@Test
public void findUserByUsername() {
    // Generování třídy user
    final User user = Generator.generateUser();
    // Nastavení moku, kdy po zavolání metody findByUsername(),
    // mok vrátí námi vygenerovaného usera
    when(serviceMock.findByUsername(user.getUsername()))
        .thenReturn(user);
    // Srovnání výledků
    assertEquals(
        user,
        serviceMock.findByUsername(user.getUsername())
    );
    }
```

Obrázek 6.1: Jednotkový test pro nalezení uživatele pomocí jeho jména

6.1.2 Integrační testy

Hlavním cílem bylo na serveru otestovat aplikační vrstvu. K tomu jsem použil integrační testy, které se zabývají testováním funkcí mezi jednotlivými komponentami. V mém případě to byly hlavně funkce CRUD (Create, Read, Update, Delete) a získávání dat z knihovny ACTS s následným zpracováním.

Příklad na obrázku 6.2 znázorňuje integrační test pro ověření funkčnosti přidání nového projektu.

```
@Test
public void addProjectToUser() {
    final User user = Generator.generateUser();
    em.persist(user);
    Project project = Generator.generateProject();
    // Zavolání metody pro uložení projektu
    service.addProjectToWorkspace(user.getId(),project);
    // Získání projektu z perzistentní vrstvy
    Project projectResult
        = em.find(Project.class,project.getId());
    assertEquals(projectResult, project);
    assertEquals(projectResult.getUser(), user);
    }
```

Obrázek 6.2: Integrační test pro přidání nového projektu

6.2 Testování uživatelského rozhraní

Pro otestování uživatelského rozhraní jsem využil techniku uživatelského testování. Jedná se o přímou interakci uživatele se systémem s cílem najít v systému chyby. Tento způsob je velmi rozšířený a také jeden z nejdůležitějších, neboť řadu chyb programátoři nejsou schopni odchytit, protože vědí jak systém funguje. Respektive programátoři ani nezkusí některou věc v systému, protože se domnívají, že tato funkce nemůže způsobit chybu, či nějak narušit běh systému. Běžný uživatel tuto skutečnost nezná, a tudíž dokáže omylem nalézt i tyto chyby. Uživatelské testování lze rozdělit do několika odvětví, já zde uvedu dvě nejčastěji používané:

- **Úplná volnost testování**
  Při tomto způsobu testování uživatel nahodile, či podle jeho uvážení, proklikává systém a hledá chybu. Uživateli je sice nastíněn účel aplikace, ale již mu nejsou řečeny všechny podrobnosti. Výhodou tohoto přístupu je snadná počáteční příprava a odhalení chyb, které nejsou znatelné na první pohled. Nevýhodou je ve většině případů špatná reprodukce[2] chyby a možnost, že uživatel nepokryje důležité části aplikace.

- **Testování pomocí předem vytvořených scénářů**
  Uživateli je vytvořeno několik testovacích scénářů, které musí projít a zkontrolovat jestli proběhlo vše jak bylo ve scénáři napsáno. Tuto metodu jsem využil ve své práci a v další části ji rozeberu.

6.2.1 Testování pomocí vytvořených scénářů

Metodika se zakládá na vytvoření scénářů o konkrétních krocích, kdy je znám a očekáván konečný stav aplikace po provedení jednotlivých úkonů. Je potřeba, aby scénáře byly jasné a detailně popsané. Každý scénář by měl obsahovat:

- **ID**
  Určuje jednoznačné označení scénáře.

- **Účel**
  Za jakým účelem je tento test prováděn.

- **Prioritu**
  Značení pomocí slov vyšší, střední, nižší k určení důležitosti akce v rámci aplikace, jinak řečeno jaký vliv daná akce má na chod systému.

- **Vstupy**
  Specifikace vstupů, jakožto i stav, ve které se aplikace na začátku testu nachází.

[2] Snaha o znovu vytvoření stejné chyby.

- **Aktivitu**
  Seznam jednotlivých kroků, které jsou pořeba provést, aby se došlo k očekávanému výsledku.
- **Očekávaný výsledek**
  Očekávaný stav aplikace po provedení všech kroků.

Uvedu zde dva příklady testovacích scénářů. První se zabývá úspěšným vytvořením nového projektu 6.1 a druhý chybou při registraci, kdy uživatelské jméno již je v databázi a uživatel si musí zvolit jiné 6.2.

ID	1
Účel	Kontrola, že uživatel je schopen vytvořit nový projekt a následně je přesměrován na stránku s nově vytvořeným projektem
Priorita	Vyšší
Vstupy	Uživatel je přihlášen, uživatel je na obrazovce se seznamem projektů
Aktivita	Uživatel stiskne tlačítko "New Project"
Očekávaný výsledek	Systém vytvoří projekt a přesměruje uživatele na obrazovku s projektem. Během vytvoření se neobjeví žádné modální okno, které by symbolizovalo problém

Tabulka 6.1: Testovací scénář č. 1

ID	2
Účel	Kontrola, že se uživateli při registraci zobrazí chybová hláška, která oznamuje, že uživatelské jméno je již v databázi a musí si zvolit jiné
Priorita	Nižší
Vstupy	Uživatel je nepřihlášen, v databázi existuje již uživatel se jménem "testovaciuzivatel", uživatel se vyskytuje na obrazovce pro registraci
Aktivita	Uživatel vyplní do políčka username toto jméno "testovaciuzivatel", zvolí si nějaké heslo delší než 6 znaků, klikne na tlačítko sign up
Očekávaný výsledek	Systém nového uživatele nezaregistruje a zobrazí uživateli varovnou hlášku: "Error: Username is already taken!"

Tabulka 6.2: Testovací scénář č. 2

Kapitola 7

Závěr

Cílem této bakalářské práce bylo vytvoření uživatelského rozhraní pro platformu DataGrid. Pan Václav Rechtberger vytvořil projekt Combinatorial testing tool, který obsahoval jak serverovou část, tak i základní neintuitivní uživatelské rozhraní. Platforma sloužila pro správu a generování testovacích scénářů, nicméně práce v uživatelském rozhraní byla náročná a nepřehledná. Zároveň systém pracoval pouze s jednou strategií pro výpočet testovací sady a nedával možnost srovnání s ostatními strategiemi.

Mým úkolem bylo tyto nedostatky odstranit a rozšířit stávající systém o další strategie. Při tvorbě rozhraní se vyskytl problém s rozšiřitelností aplikace. Aplikace byla napsána v jazyku Java a pro zobrazení uživatelského rozhraní využívala syntaxi JSX. Tento princip mi nedával možnost oddělit uživatelské rozhraní, u kterého jsem chtěl použít externí knihovny a framework React.js.

Aplikace tak musela být přepsána do architektury rest-api. K úpravě jsem využil framework Spring Boot, se kterým jsem již měl nějaké zkušenosti, a proto mi přepis netrval příliš dlouho. Pro autorizaci a autentizaci jsem zvolil standardu JSON Web Token, který je populárním řešením tohoto problému. Díky tomuto úkonu se systém rozdělil na serverovou část a uživatelské rozhraní.

Pro uživatelské rozhraní jsem zvolil framework React.js s několika rozšiřujícími knihovnami. Z knihoven stojí za zmínku knihovna Redux (pro ukládání stavu aplikace do centrálního skladu) a knihovna Chart.js pro zobrazení grafu, který zobrazoval srovnání výsledků vybraných algoritmů.

Po analýze a přepisu serverové části již bylo snadné propojit uživatelskou a serverovou část aplikace. Vytvořil jsem uživatelské rozhraní pro tvorbu a správu projektu, což byl hlavní úkol mé bakalářské práce. Především jsem kladl důraz na vizuální stránku aplikace, jelikož pro velké kombinatorické projekty se zobrazení dat stává nepřehledné a uživatel by mohl snáze udělal chybu. Zároveň jsem nechtěl uživatele zatěžovat zbytečnými úkony, které je možné provádět na pozadí, jako je například ukládání prvků do systému. Součástí aplikace jsou i v dnešní době stále více populární modální okna a včasné informování uživatele o chybě.

Požadavky ze zadání bakalářské práce jsou splněny a důkazem jsou úspěšné průchody testovacích scénářů. Při testování se objevilo několik chyb, které měly jen minoritní dopad na chod aplikace, přesto byly tyto chyby opraveny

a znovu otestovány. Aplikace je funkční a lze ji používat na účely, pro které byla vytvořena. Srovnání vybraných algoritmů je přehledné, vygenerovaný graf se srovnáním je dostatečně názorný a je zde i úprava designu pro zařízení s malým rozlišením. Vybrané technologie, které jsou použité v aplikaci, mají dobré vlastnosti k rozšiřitelnosti systému, či jeho úpravě.

Vhodné další rozšíření aplikace by byla kooperace více uživatelů v rámci jednoho projektu (například i práce na jednom projektu v reálném čase), zobrazení sady parametrů do interaktivních diagramů, intuitivní přizpůsobení na mobilní zařízení, či našeptávání hodnot při vyplňování polí. Tyto funkce nebyly součástí zadání mé bakalářské práce a jedná se tedy jen o mé návrhy k budoucímu vývoji aplikace.

Bakalářská práce byla pro mě velmi přínosná, z hlediska vývoje aplikace jako celku. Dnešní firmy rozdělují aplikace do několika vrstev, kdy programátor většinou pracuje pouze jen v jedné vrstvě, a tudíž nevidí do celé aplikace. Tento přístup je nezbytný v případě velkých projektů, avšak je velmi přínosné, z hlediska zkušeností, si zkusit vybudovat celý systém od začátku. Toto jsem si zde vyzkoušel, jelikož jsem musel přepisovat serverovou část a pracovat ve všech vrstvách aplikace, tj. od databáze přes aplikační vrstvu až po graficky přívětivé uživatelské rozhraní. Využil jsem řadu známých frameworků, čímž jsem si prohloubil znalosti o jejich možnostech a samotný projekt, který je zaměřen na kombinatorické testování, mě také velmi nadchnul a rád bych využíval principy a strategie zde popsané.

Literatura

[1] *D. Richard Kuhn, Raghu N. Kacker, Yu Lei,* **Introduction to Combinatorial Testing.**
Chapman & Hall/CRC Innovations in Software Engineering and Software Development Series, 2013

[2] *Miroslav Bures, Bestoun S. Ahmed,* **On the Effectiveness of Combinatorial Interaction Testing: A Case Study.**
Proceedings of 2017 IEEE International Conference on Software Quality, Reliability and Security (Companion Volume), IEEE, p. 70-76.

[3] *Hasan, Imad H., Bestoun S. Ahmed, Moayad Y. Potrus, and Kamal Z. Zamli,* **Generation and Application of Constrained Interaction Test Suites Using Base Forbidden Tuples with a Mixed Neighborhood Tabu Search.**
International Journal of Software Engineering and Knowledge Engineering, 30(03), 2020, pp. 363-398.

[4] *Jan Richter, Bestoun S. Ahmed, Miroslav Bures and Cleber R. Rosa Junior,* **Avocado: Open-Source Flexible Constrained Interaction Testing for Practical Application.**
Accepted at IWCT 2020, part of the IEEE International Conference on Software Testing, Verification and Validation (ICST) 2020.

[5] *Zamli, K. Z., Din, F., Ahmed, B. S., Bures, M.,* **A hybrid Q-learning sine-cosine-based strategy for addressing the combinatorial test suite minimization problem.**
PloS one, 13(5), 2018, e0195675.

[6] *M. Bureš, M. Renda, M. Doležel et al.,* **Efektivní testování softwaru: Klíčové otázky pro efektivitu testovacího procesu.** Grada Publishing as, 2016.

[7] *Petr Roudenský, Anna Havlíčková,* **Řízení kvality softwaru.** 2017, ISBN 978-80-251-4519-7.

[8] *Bestoun S. Ahmed, Angelo Gargantini, Kamal Z. Zamli, Cemal Yilmaz, Miroslav Bures, Marek Szeles,* **Code-Aware Combinatorial Interaction Testing.** IET Software, 13(6), pp. 600-609, 2019.

[9] *Bestoun S. Ahmed, Luca M. Gambardella, Wasif Afzal, and Kamal Z. Zamli,* **Handling Constraints in Combinatorial Interaction Testing in the presence of Multi Objective Particle Swarm and Multi-threading.** Information and Software Technology, Vol (86), Pages 20-36, (2017), Elsevier.

[10] *Jian Zhang, Zhiqiang Zhang, Feifei Ma,* **Automatic Generation of Combinatorial Test Data.** 2014, ISBN 978-3-662-43429-1

[11] *ProfessionalQA,*
Combinatorial Testing, Feb 24, 2020. [Online]. Available: http://www.professionalqa.com/combinatorial-testing

[12] *bezkoder,*
Spring Boot + React: JWT Authentication with Spring Security, May 13, 2020. [Online]. Available: https://bezkoder.com/spring-boot-react-jwt-auth/

[13] *Metodika testování podle mezinárodních praktik a standardů,*
Testovací případ. [Online]. Available: http://metodikamezitest.asp2.cz/Metodika_testovani/workproducts/testovaci_pripad_85F09AB7.html

[14] *Yu Lei, Raghu Kacker, D. Richard Kuhn, Vadim Okun, James Lawrence,*
IPOG: A General Strategy for T-Way Software Testing, May 13, 2020. [Online]. Available: https://tsapps.nist.gov/publication/get_pdf.cfm?pub_id=50944

[15] *Yu Lei, Raghu Kacker, D. Richard Kuhn, Vadim Okun and James Lawrence,*
IPOG/IPOG-D: efficient test generation for multi-way combinatorial testing, 29 November 2007 . [Online]. Available: https://tsapps.nist.gov/publication/get_pdf.cfm?pub_id=50944

[16] *Mohammed I. Younis and Kamal Z. Zamli,*
MIPOG - An Efficient t-Way Minimization Strategy for Combinatorial Testing, International Journal of Computer Theory and Engineering, Vol. 3, No. 3, June 2011. [Online]. Available: https://pdfs.semanticscholar.org/3b26/21faf005d7391eff506f31d329ce67091425.pdf

[17] *Tomáš Hlava,*
Fáze a úrovně provádění testů, 21.8.2011. [Online]. Available: http://testovanisoftwaru.cz/tag/integracni-testovani/

[18] *Tutorialspoint,*
Spring Boot - Introduction. [Online]. Available: https://www.tutorialspoint.com/spring$_b$oot/spring$_b$oot$_i$ntroduction.htm

[19] *Good Rebels,*
To go or not to go micro: the pros and cons of microservices. Aug 16, 2018. [Online]. Available: https://medium.com/@goodrebels/to-go-or-not-to-go-micro-the-pros-and-cons-of-microser

[20] *Journaldev,*
Spring Boot Tutorial. [Online]. Available: https://www.journaldev.com/7969/spring-boot-tutorial

[21] *Ondřej Váško,*
Top 10 frameworků pro moderní frontend. 10.2.2017. [Online]. Available: https://www.wdt.cz/inovace/top-10-frameworku-pro-moderni-frontend-a589ddee

[22] *Shaumik Daityari,*
Angular vs React vs Vue: Which Framework to Choose in 2020. July 29, 2020. [Online]. Available: https://www.codeinwp.com/blog/angular-vs-vue-vs-react/

[23] *Piero Borrelli,*
Angular vs. React vs. Vue: A performance comparison. August 29, 2019. [Online]. Available: https://blog.logrocket.com/angular-vs-react-vs-vue-a-performance-comparison/

[24] *Sandy Weck,*
Developing modern offline apps with ReactJS, Redux and Electron - Part 3 - ReactJS + Redux. 12.03.2017. [Online]. Available: https://blog.codecentric.de/en/2017/12/developing-modern-offline-apps-reactjs-redux-e

[25] *James Bach and Patrick J. Schroeder,*
Pairwise Testing: A Best Practice That Isn't. 2004. [Online]. Available: http://www.testingeducation.org/wtst5/PairwisePNSQC2004.pdf

[26] *Scott W. Ambler,*
Examining the Agile Cost of Change Curve. 2004. [Online]. Available: http://www.agilemodeling.com/essays/costOfChange.htm

[27] *Shivaprakash,*
ReactJS Tutorial - Design Your Web UI Using ReactJS JavaScript Library. Apr 29. 2020. [Online]. Available: https://www.edureka.co/blog/reactjs-tutorial

Příloha A

Testovací scénáře

ID	1
Účel	Kontrola, že uživatel je schopen vytvořit nový projekt a následně je přesměrován na stránku s nově vytvořeným projektem
Priorita	Vyšší
Vstupy	Uživatel je přihlášen, uživatel je na obrazovce se seznamem projektů
Aktivita	Uživatel stiskne tlačítko "New Project"
Očekávaný výsledek	Systém vytvoří projekt a přesměruje uživatele na obrazovku s projektem, během vytvoření se neobjeví žádné modální okno, které by symbolizovalo problém

Tabulka A.1: Testovací scénář č. 1

ID	2
Účel	Kontrola, že se uživateli při registraci zobrazí chybová hláška, která oznamuje, že uživatelské jméno je již v databázi a musí si zvolit jiné
Priorita	Nižší
Vstupy	Uživatel je nepřihlášen, v databázi existuje již uživatel se jménem "testovaciuzivatel", uživatel se vyskytuje na obrazovce pro registraci
Aktivita	Uživatel vyplní do políčka username toto jméno "testovaciuzivatel", zvolí si nějaké heslo delší než 6 znaků, klikne na tlačítko "Sign Up"
Očekávaný výsledek	Systém nového uživatele nezaregistruje a zobrazí uživateli varovnou hlášku: "Error: Username is already taken!"

Tabulka A.2: Testovací scénář č. 2

ID	3
Účel	Kontrola, že uživatel je schopen se registrovat
Priorita	Vyšší
Vstupy	Uživatel není přihlášen, uživatel je na obrazovce s registrací
Aktivita	Uživatel do pole "Username" napíše libovolné slovo, které bude delší než 3 znaky a kratší než 20 znaků a ještě není zaregistrované v systému, následně uživatel vyplní od pole "Password" libovolné slovo, které bude mít délku v rozmezí [6-40], poté klikne na tlačítko registrovat
Očekávaný výsledek	Systém zaregistruje nového uživatele a přihlásí ho do aplikace, uživateli se zobrazí úvodní obrazovka aplikace

Tabulka A.3: Testovací scénář č. 3

ID	4
Účel	Kontrola vytvoření nové sady parametrů
Priorita	Vyšší
Vstupy	Uživatel je přihlášen, uživatel je na obrazovce s vytvořeným projektem
Aktivita	Uživatel stiskne tlačítko "New Parameter Set"
Očekávaný výsledek	Systém vytvoří novou sadu parametrů, která se zobrazí uživateli, tato sada nebude obsahovat žádný parametr a při obnovení stránky bude nová sada parametrů stále přítomna v projektu (důkaz, že se uložila do databáze)

Tabulka A.4: Testovací scénář č. 4

ID	5
Účel	Kontrola, že systém odstraní sadu parametrů
Priorita	Střední
Vstupy	Uživatel je přihlášen, uživatel má vytvořen projekt a je na obrazovce s tímto projektem, uživatel má vytvořenou alespoň jednu sadu parametrů
Aktivita	Uživatel stiskne tlačítko "Remove" v oblasti sady parametrů
Očekávaný výsledek	Systém smaže sadu parametrů, odstraní ji vizuálně z projektu a uživateli se nezobrazí žádné modální okno s problémem

Tabulka A.5: Testovací scénář č. 5

ID	6
Účel	Kontrola, že systém zobrazí uživateli graf se srovnáním algoritmů
Priorita	Střední
Vstupy	Uživatel je přihlášen, uživatel má vytvořený projekt, je na obrazovce projektu, má vytvořenou alespoň jednu sadu parametrů, má vytvořen alespoň jeden parametr a u každého parametru má minimálně vytvořenou jednu hodnotu
Aktivita	Uživatel stiskne tlačítko "Graph"
Očekávaný výsledek	Systém zobrazí uživateli modální okno s grafem, který znázorňuje porovnání jednotlivých algoritmů

Tabulka A.6: Testovací scénář č. 6

ID	7
Účel	Kontrola, že uživatel je schopen vytvořit novou testovací sadu
Priorita	Vyšší
Vstupy	Uživatel je přihlášen, uživatel má vytvořený projekt, je na obrazovce projektu, má vytvořenou alespoň jednu sadu parametrů, má vytvořen alespoň jeden parametr a u každého parametru má minimálně vytvořenou jednu hodnotu
Aktivita	Uživatel stiskne tlačítko "Generate uniform strength test set", uživateli se zobrazí modální okno pro nastavení generování testovací sady, uživatel zvolí libovolné jméno, nastaví sílu pokrytí pomocí posuvného tlačítka a zvolí algoritmus, se kterým se má testovací sada vygenerovat, následně klikne na tlačítko "Generate"
Očekávaný výsledek	Systém vytvovří novou testovací sadu ze zadaných parametrů a přesměruje uživatele na obrazovku s výsledky testovací sady

Tabulka A.7: Testovací scénář č. 7

ID	8
Účel	Kontrola, že uživatel je schopen vytvořit konfiguraci pro projekt
Priorita	Nižší
Vstupy	Uživatel je přihlášen, uživatel má vytvořený projekt, je na obrazovce projektu, má vytvořenou alespoň jednu sadu parametrů, má vytvořen alespoň jeden parametr a u každého parametru má minimálně vytvořenou jednu hodnotu
Aktivita	Uživatel stiskne tlačítko "Manage configurations"u sady parametrů, po zobrazení obrazovky uživatel klikne na tlačítko "Create new configuration", uživatel klikne na tlačítko "Create interaction set", uživatel zvolí jméno sady interakcí, sílu pokrytí a naklikne na tlačítka parametrů v sekci "Select parameters"
Očekávaný výsledek	Systém vytvoří novou konfiguraci pro sadu parametrů, uživateli se nezobrazí chybová hláška a při obnovení obrazovky, bude konfigurace stále přítomna

Tabulka A.8: Testovací scénář č. 8

ID	9
Účel	Kontrola, že uživatel je schopen se odhlásit
Priorita	Střední
Vstupy	Uživatel je přihlášen
Aktivita	Uživatel stiskne tlačítko "Log out"v levém horním rohu obrazovky
Očekávaný výsledek	Systém uživatele odhlásí a zobrazí uživateli obrazovku pro přihlášení

Tabulka A.9: Testovací scénář č. 9

ID	10
Účel	Kontrola, že uživatel je schopen si prohlédnout a seřadit seznam projektů
Priorita	Nižší
Vstupy	Uživatel je přihlášen, uživatel je na úvodní obrazovce, uživatel má již vytvořené minimálně 3 projekty
Aktivita	Uživatel stiskne tlačítko "Projects"a to buď na úvodní stránce nebo v pravém horním rohu obrazovky, po zobrazení tabulky uživatel klikne na šipky vedle názvu sloupce "Project ID"
Očekávaný výsledek	Systém zobrazí uživateli seznam projektů v podobě tabulky a po kliknutí na šipky u sloupce "Project ID" se projekty seřadí vzestupně, či sestupně podle id projektu

Tabulka A.10: Testovací scénář č. 10

ID	11
Účel	Kontrola, že uživateli je zobrazena chybová hláška, pokud chce vytvořit testovací sadu s prázdným omezením
Priorita	Střední
Vstupy	Uživatel je přihlášen, uživatel má vytvořený projekt, má vytvořenou alespoň jednu sadu parametrů, má vytvořen alespoň jeden parametr, u každého parametru má minimálně vytvořenou jednu hodnotu, uživatel je na obrazovce konfigurace sady parametrů a má vytvořenou alespoň jednu konfiguraci, uživatel nemá vytvořenou sadu interakcí
Aktivita	Uživatel stiskne tlačítko "Create test set with configuration"
Očekávaný výsledek	Systém zobrazí uživateli hlášku o potřebě mít v konfiguraci alespoň jednu sadu interakcí a nezobrazí modální okno pro generování testovací sady

Tabulka A.11: Testovací scénář č. 11

ID	12
Účel	Kontrola, že uživateli je zobrazena chybová hláška při pokusu vytvoření testovací sady s vytvořenou konfigurací a žádnou sadou interakcí
Priorita	Střední
Vstupy	Uživatel je přihlášen, uživatel má vytvořený projekt, má vytvořenou alespoň jednu sadu parametrů, má vytvořen alespoň jeden parametr, u každého parametru má minimálně vytvořenou jednu hodnotu. Je na obrazovce konfigurace sady parametrů a má vytvořenou alespoň jednu konfiguraci a jednu sadu interakcí
Aktivita	Uživatel stiskne tlačítko "Create constraint", do vytvořeného pole nic nevyplní a klikne na tlačítko "Create test set with configuration"
Očekávaný výsledek	Systém zobrazí uživateli hlášku o problému s prázdnou hodnotou omezení a nezobrazí modální okno pro generování testovací sady

Tabulka A.12: Testovací scénář č. 12