VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INTELIGENTNÍCH SYSTÉMŮ
DEPARTMENT OF INTELLIGENT SYSTEMS

UŽIVATELSKÉ ROZHRANÍ NÁSTROJE COMBINE

USER INTERFACE FOR TOOL COMBINE

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **PATRIK NENUTIL**
AUTHOR

VEDOUCÍ PRÁCE **Ing. ALEŠ SMRČKA, Ph.D.**
SUPERVISOR

BRNO 2024

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

Zadání bakalářské práce

155693

Ústav: Ústav inteligentních systémů (UITS)
Student: **Nenutil Patrik**
Program: Informační technologie
Název: **Uživatelské rozhraní nástroje Combine**
Kategorie: Analýza a testování softwaru
Akademický rok: 2023/24

Zadání:

1. Nastudujte kombinační testování. Nastudujte aktuální verzi nástroje Combine vyvíjeném na FIT VUT.
2. Analyzujte požadavky na použití nástroje Combine přes uživatelské rozhraní a přes RestAPI. Navrhněte uživatelské a programové rozhraní nástroje Combine.
3. Implementujte uživatelské rozhraní nástroje Combine.
4. Navrhněte a implementujte automatické GUI testy vaší implementace.

Literatura:

- Kacker, R. (2013), An Efficient Algorithm for Constraint Handling in Combinatorial Test Generation, Proceedings of Sixth IEEE International Conference on Software Testing, Verification and Validation ICST 2013, Luxembourg, -1, [online], https://tsapps.nist.gov/publication/get_pdf.cfm?pub_id=913191
- Nástroj Combine, https://pajda.fit.vutbr.cz/testos/combine-bcc zdrojové kódy dostupné na: https://pajda.fit.vutbr.cz/testos/combine-bcc

Při obhajobě semestrální části projektu je požadováno:
První dva body zadání.

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce: **Smrčka Aleš, Ing., Ph.D.**
Vedoucí ústavu: Hanáček Petr, doc. Dr. Ing.
Datum zadání: 1.11.2023
Termín pro odevzdání: 9.5.2024
Datum schválení: 6.11.2023

Abstrakt

Cílem této bakalářské práce je vytvořit nové uživatelské rozhraní pro nástroj Combine, protože stávající bylo vytvořeno pouze jako prototyp pro ukázku práce s nástrojem. Nově vytvořené uživatelské rozhraní umožňuje využívání datových sad a zjednodušených zápisů kritérií. Také byly opraveny nedostatky podle nastudovaných principů. Pro kontrolu správnosti implementace byly vytvořeny automatické testy grafického uživatelského rozhraní. Výsledkem této práce je vylepšené uživatelského rozhraní pro nástroj Combine.

Abstract

This bachelor's thesis aims to create a new user interface for the Combine tool, since the current one was merely a prototype to showcase the tool's operations. The newly developed user interface allows use of datasets and simplified criteria inserting. Furthermore, shortcomings were fixed based on the studied principles. Automated GUI tests were implemented to validate the implementation's accuracy. As a result of this project, the user interface of tool Combine was improved.

Klíčová slova

kombinační testování, uživatelské rozhraní, aplikační rozhraní, GUI, React

Keywords

combinatorial testing, user interface, application interface, GUI, React

Citace

NENUTIL, Patrik. *Uživatelské rozhraní nástroje Combine.* Brno, 2024. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Aleš Smrčka, Ph.D.

Uživatelské rozhraní nástroje Combine

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Aleše Smrčky, Ph.D. Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....
Patrik Nenutil
7. května 2024

Poděkování

Rád bych poděkoval vedoucímu mé bakalářské práce, panu Ing. Aleši Smrčkovi, Ph.D. za jeho přínosné vedení, strávený čas, a užitečné informace, které mi pomohly při tvorbě této bakalářské práce.

Obsah

Seznam obrázků

Kapitola 1

Úvod

Cílem této práce je vylepšit uživatelské rozhraní pro již existující nástroj *Combine*, který byl vyvinut na Fakultě informačních technologií Vysokého učení technického v Brně v rámci bakalářské práce. Společně s tímto nástrojem bylo vytvořeno také webové uživatelské rozhraní, ale to slouží pouze jako prototyp pro ukázku práce s nástrojem. *Combine* slouží pro generování testovacích dat za využití kombinačního testování.

Před samotným vytvořením uživatelského rozhraní byl vytvořen návrh, ve kterém se nachází analýza nalezených nedostatků v původní verzi a jejich možné řešení. Mezi hlavní nedostatky patří špatná práce se vstupy a nedostatečné usnadnění práce uživateli. Návrh také obsahuje seznam uživatelských požadavků, které musí být splněny, aby mohla být nová verze považována za vylepšenou.

Nově implementované řešení uživatelského rozhraní zachovává funkcionalitu předešlé implementace, ale navrch přináší i nové funkcionality. Mezi ty patří možnost exportování a importování nastavení, výběr hodnot bloků parametrů z předdefinovaných datových sad a nebo výběr hodnot bloků pomocí kritérií.

Uživatelské rozhraní bylo po vytvoření otestováno automatickou sadou testů grafického uživatelského rozhraní. Touto testovací sadou byly otestovány všechny základní části aplikace. Po provedení všech testů byly opraveny všechny nalezené chyby.

Kapitola 2

Současný stav nástroje

Tato kapitola má za cíle seznámit čtenáře s problematikou uživatelských rozhraní v podkapitole 2.1. Dále se zabývá teorií kombinačního testování kvůli pochopení dalších kapitol v podkapitole 2.2, protože znalost kombinačního testování je důležitá pro pochopení návrhu uživatelského rozhraní nástroje Combine. Poslední podkapitola 2.3 popisuje aktuální verzi nástroje Combine.

2.1 Uživatelské rozhraní

> Uživatelské rozhraní interaktivních systémů je styčným bodem člověka s informačními a komunikačními technologiemi (ICT). Jako lidský výtvor je součástí kultury, která působí a ovlivňuje nás, často aniž si to plně uvědomujeme. Uživatelské rozhraní usměrňují interakci tak, aby podporovala záměr uživatele. Často ale více podléhá záměru a znalostem zadavatele, designéra nebo toho, co systém umožní a dochází tak k manipulaci.

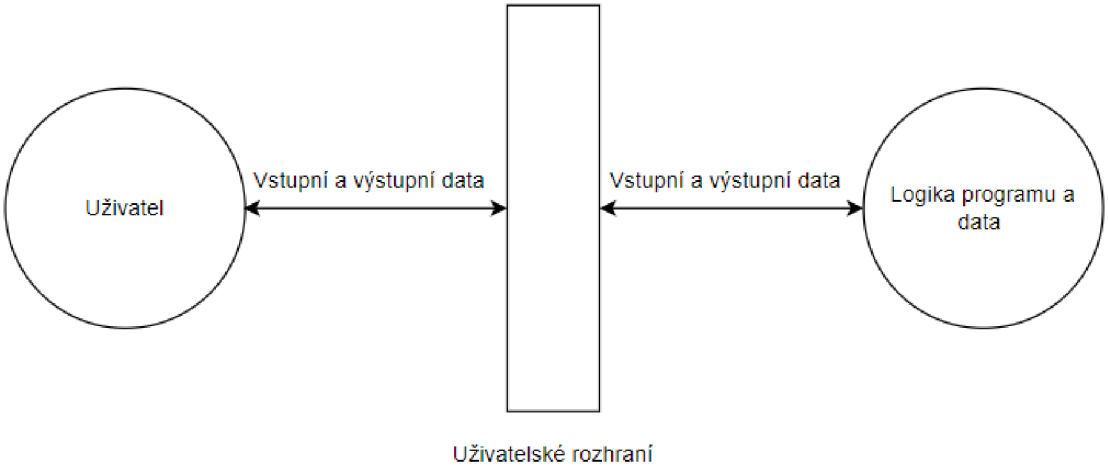
Tato pasáž byla převzata z [1].

Uživatelské rozhraní tedy chápeme jako rozhraní mezi člověkem a počítačem, které slouží k výměně informací skrze vstupy a výstupy. Ty představují cestu toku dat od uživatele do programu, který má za cíl data zpracovat, vyhodnotit a uživateli zpřístupnit požadovaný výsledek. Tento proces lze vidět na obrázku 2.1.

Podstatnou částí uživatelských rozhraní je uživatelská zkušenost (anglicky user experience, zkráceně UX). Uživatelská zkušenost se zabývá veškerými aspekty interakce uživatele s daným produktem. Čím lepší je uživatelská zkušenost, tím lepší je celé uživatelské rozhraní. Pokud je uživatelská zkušenost dobrá, tak je větší pravděpodobnost, že uživatel bude chtít takový produkt používat. Dánský odborník v oblastní designu Jakob Nielsen vytvořil příručku deseti principů, které by měly uživatelská rozhraní splňovat, aby byla zaručena použitelnost aplikace. Tyto principy jsou popsány v podkapitole 2.1.1.

2.1.1 Principy použitelnosti uživatelského rozhraní

Tato podkapitola popisuje jednotlivé principy použitelnosti. S použitelným uživatelským rozhraním by měli uživatelé pracovat rádi, musí být snadné se v něm zorientovat a práce v něm by se měla dát rychle naučit. Informace byly čerpány z [3]. Praktická ukázka analýzy těchto principů a jejich nesplnění v aktuálním uživatelském rozhraní nástroje Combine se nachází v kapitole 3.2.



Obrázek 2.1: Diagram práce uživatelského rozhraní

1. Viditelnost stavu systému

Viditelnost stavu systému znamená, že uživatel musí být informován o právě probíhajících akcích, a to co nejpřehlednější cestou. Další věcí, kterou by měl uživatel dostat je zpětná vazba, která se váže na jeho pokusy o nějakou operaci. Například pokud uživatel klikne na tlačítko, které má uložit data, tak by měl být informován o úspěchu nebo neúspěchu operace.

2. Spojení mezi systémem a reálným světem

Tento princip je o tom, že uživatelské rozhraní by mělo s uživatelem komunikovat pomocí známých a zavedených prostředků a mělo by být přizpůsobené přirozeným zvyklostem, které uživatelé automaticky očekávají. Tím je myšleno, že uživatelské rozhraní využívá:

- Zavedené ikony, které se uživateli spojí se danou operací. Například operace uložit = ikona diskety nebo operace smazání = ikona koše.
- Srozumitelné hlášky, které musí využívat takových slov, aby uživatel nemusel vyhledávat jejich význam.

3. Uživatelská kontrola a svoboda

Tento princip hovoří o tom, že pokud uživatel provede nějakou operaci omylem, měl by být schopen jednoduchým způsobem operaci navrátit do předchozího stavu. Uživatel si pak bude připadat bezpečněji, protože ví, že když udělá omyl, tak je schopen chybu snadno opravit. Prakticky tento princip znamená, že by rozhraní mělo nabízet operace pro vrácení operace a možnost zrušit aktuálně probíhající operaci.

4. Konzistence a standardizace

Princip konzistence a standardizace spočívá v tom, že prvky v aplikaci by měly být zachovány ve stejné podobě na všech místech. Zde se jedná o dva druhy konzistence. První je interní konzistence, která znamená, že prvky aplikace, které jsou stejné nebo velice podobné

by měly vypadat a chovat se stejně v celé aplikaci. Druhá je externí konzistence. Uživatel má již určitě zkušenosti s používáním jiných uživatelských rozhraní, které používají stejných postupů a očekává je i v ostatních aplikacích. V takovém případě je přechod mezi více uživatelskými rozhraními jednodušší.

5. Prevence chyb

V tomto principu se jedná o to, že pokud je možné chybě předejít, tak je to nejlepší varianta. Mezi takové chyby patří buď chyba z nepozornosti, nebo taková chyba, kde uživatel nepochopil uživatelské rozhraní. Chyby z nepozornosti se dají řešit například vyžádáním potvrzení operace nebo kontrolou vstupů. Dalším vhodným prvkem je napovídání vstupních hodnot uživateli.

6. Rozpoznání místo vzpomínání

Tento princip doporučuje, aby uživatelé byli schopni snadno rozpoznat co mají v uživatelském rozhraní dělat, místo toho, aby spoléhali na svou paměť. Klade se zde důraz na to, aby byl různé akce v aplikaci reprezentovány způsobem, který uživateli indikuje jeho možnosti a které akce může provést. Aby byl tento princip zajištěn, tak je vhodné využívat srozumitelné a pochopitelné ikony. Dále je třeba zachovat konzistenci mezi různými prvky.

7. Flexibilní a efektivní použití

Tento princip hovoří o tom, že uživatelské rozhraní by mělo umožňovat pokročilým uživatelům používat funkcionality, které jim práci urychlí, protože už vědí, co mají dělat. Tyto funkcionality však nesmí znemožňovat práci novým uživatelům. Mezi takové funkcionality patří například využívání klávesových zkratek pro často opakované akce. Jedním z příkladů tohoto principu je taky možnost uživatele si přizpůsobovat toto rozhraní tak, aby uživateli vyhovovalo.

8. Estetický a minimalistický vzhled

Uživatelské rozhraní by nemělo obsahovat takové části, které je pro uživatele zbytečné a mělo by se omezit pouze na elementy, prvky a informace, které jsou v daný okamžik pro uživatele podstatné. Minimalistický by měl zůstat i grafický vzhled rozhraní. Tento přístup pak lépe zvýrazní důležité prvky a uživatel se může soustředit na správný obsah.

9. Pomoc uživatelům s chybami

Pokud se v určité části rozhraní nepodaří splnit princip "Prevence chyb" 2.1.1, pak je vhodné dle tohoto principu uživateli poskytnou zpětnou vazbu v podobě výstižné chybové hlášky, ze které uživatel ihned rozpozná, jaké kroky musí podstoupit, aby nastalou chybu vyřešil. Chybové hlášky by také měly být v jazyce, který je pro uživatele srozumitelný. Nesmí se v nich tedy vyskytovat technické informace o chybě. Dále by měly být chybové hlášky zabarveny v tradiční červené barvě, která indikuje chybový stav.

10. Nápověda a dokumentace

Pokud uživatelské rozhraní potřebuje další informace ohledně používání, protože není samovysvětlující, tak je vhodné vytvořit k němu dokumentaci nebo nápovědu, která uživateli

objasní postupy, jak docílit požadovaných kroků. Taková nápověda a dokumentace by měla být stručná a mělo by se v ní dát snadno orientovat. Nápověda by se měla nacházet u prvků přesně v ten moment, kdy by ji uživatel mohl požadovat.

2.2 Kombinační testování

Kombinační testování je typ testování, které je účinné na odhalování chyb v software produktech. Tento typ testování se zaměřuje na pokrytí kombinací hodnot s cílem nalezení co největšího počtu chyb s co nejmenším počtem testovacích případů. Prakticky to znamená získání co největší efektivity testů, ale zároveň nízké náročnosti na výpočetní výkon. Hlavním principem kombinačního testování je, že na rozdíl od běžného přístupu, kde se testují pouze jednotlivé artefakty, se u kombinačního testování berou v úvahu kombinace a vztahy jednotlivých artefaktů, kde artefakt značí jakýkoliv prvek nebo entitu, která může být identifikována a testována.

Kombinace mohou být realizovány ručně, ale existují automatické nástroje pro vytváření testovacích sad pomocí kombinací jako právě Combine, o kterém pojednává kapitola 2.3. Tyto nástroje využívají kombinačních algoritmů, jako je třeba algoritmus *IPOG* a jeho vylepšená varianta *IPOG-C*, o kterých pojednává kapitola 2.2.2.

Není však možné vždy testovat veškeré možné kombinace hodnot, protože takový přístup by byl výpočetně velice náročný. Už jen testování funkce, která obsahuje dvě proměnné by bylo velice náročné, a dokonce až téměř nemožné. Jako příklad lze uvést testování funkce se dvěma vstupními proměnnými datového typu osmibitového čísla, tedy tato proměnná může nabývat 256 různých hodnot. Pokud by byl potřeba test pro každou možnou kombinaci, dostali bychom se k číslu $256^2 = 65536$. U každého testu by bylo nezbytné sepsat testovací scénář a přiřadit k němu očekávanou výstupní hodnotu této funkce. Testování celého programu by tímto způsobem bylo prakticky nemožné.

Cílem pro praktické využití kombinačního testování tedy je správným nastavením kombinací snížit datovou sadu tak, aby stále dokázala najít co největší množství chyb. Informace v této sekci jsou vytaženy z [9].

2.2.1 Typy kritéria pokrytí

Tato sekce ukazuje všechny typy kritérií pokrytí. Díky správnému výběru kritéria pokrytí lze měnit velikost datové sady. Tyto informace byly vytaženy z [9].

- **All Combinations Coverage** - Jak již název napovídá, toto kritérium pokrytí vyžaduje, aby byly použity všechny možné kombinace bloků. Je to tedy nejsilnější možné kritérium pokrytí, protože je požadována kombinace všech artefaktů s každým jiným.
- **Each Choice Coverage** - Na rozdíl od *All Combinations Coverage* dělá přesný opak a tedy vyžaduje, aby se blok každého oddílu ve výsledné testovací sadě vyskytl alespoň jednou. Je to tedy nejslabší možné kritérium pokrytí, protože vyžaduje jen jedno využití každého artefaktu.
- **Pair-Wise Coverage** - Toto kritérium vyžaduje pokrytí kombinací tak, aby blok každého oddílu byl zkombinován s každým blokem z oddílů ostatních.
- **T-Wise Coverage** - Toto kritérium pokrytí je podstatně pro celý nástroj Combine. Jedná se o takové kritérium, kde na rozdíl od Pair-Wise Coverage nepokrývá dvojice,

ale n-tice podle vybraného o počtu prvků podle vybraného T. T, neboli T-strength je tedy síla kombinací, které musí testovací sada pokrýt. Čím je vyšší síla T, tím více případů bude pokryto. Důležitou informací je, že síla T nemůže být větší než vstupní počet oddílů, které mají být pokryty.

- **Base Choice Coverage** - Nejdůležitějším rozdílem oproti předchozím kritériím je, že umožňuje specifikovat, který blok je nejdůležitější, a bude tedy pravděpodobněji vybrán do kombinace. Při vytváření datové sady pomocí Base Choice Coverage se vybere bázový blok a každá datová sada bude obsahovat kombinaci těchto bázových bloků.

2.2.2 Alogritmy pro kombinační testování

V této kapitole jsou obecně popsány algoritmy IPOG a IPOG-C, které slouží k automatickému vygenerování testovacích sad za pomocí poskytnutých vstupů v podobě parametrů a jejich bloků, kde bloky označují konkrétní data, která by měla být v testech zahrnuta.

IPOG

Tato sekce využívá informací ze zdroje [2]. In Parameter Order Generalization (IPOG) je algoritmus, jehož hlavní úlohou je generovat minimální množství testovacích případů, ale zároveň pokrýt všechny kombinace vstupních parametrů. Prvním vstupem tohoto algoritmu je číslo t, které určuje sílu kombinací, které musí být testovací sadou pokryty a druhým vstupem je množina parametrů obsahujících hodnoty nebo rozmezí hodnot. Z těchto parametrů pak budou vytvořeny kombinace.

IPOG-C

Tato sekce využívá informací ze zdroje [7]. In Parameter Order Generalization with Constraints je algoritmus, který v principu funguje stejně jako IPOG 2.2.2, ale přidává k němu optimalizační nadstavbu v podobě omezení. Pomocí těchto omezení lze specifikovat, které kombinace nedávají smysl a mohou být z procesu kombinování vynechány.

2.3 Nástroj Combine

Tato podkapitola se zaměřuje na nástroj Combine, určený pro testování systémů. Combine představuje webovou aplikaci i samostatný nástroj, do kterého lze vkládat parametry testovaného systému SUT, kde SUT (System under test) označuje celý systém, který podléhá testování. Za využití vstupních parametrů nástroj umožňuje vygenerovat testovací data, která pokrývají všechny T-násobné kombinace bloků vložených parametrů. Parametry jsou definovány identifikátorem, datovým typem a bloky, které představují konkrétní prvky, kterých parametr může nabývat. Do nástroje lze vkládat parametry až deseti datových typů a T-Strentgh (síla T) pro specifikaci, jak velké má být pokrytí výskytu jednotlivých bloků, které mohou být až šesticemi, v testovacích datech.

Testovací data lze generovat s indexy bloků, nebo s konkrétními hodnotami, které jsou nástrojem vygenerovány. V případě kombinací, které z logického hlediska nedávají smysl a neměly by být ve výsledných testovacích datech zahrnuty, lze specifikovat omezení (anglicky Constraints) některých kombinací. Výsledná testovací data může uživatel stáhnout ve formátech XML, CSV a JSON. Tato podkapitola je převzata z [9].

Kapitola 3

Návrh rozhraní

Tato kapitola se zaměřuje na návrh uživatelského a aplikačního rozhraní pro nástroj *Combine*. Jako první je potřeba zmínit, že pro správné pochopení celého nástroje bylo využito zdroje [9] a uživatelského manuálu aktuální verze nástroje [8]. Nejdříve je vhodné analyzovat uživatelské požadavky, které jsou shrnuty v podkapitole 3.1. Následně se zde nachází shrnutí všeobecných nedostatků aktuálního rozhraní v podkapitole 3.2. Dále výběr řešení těchto nedostatků v podkapitole 3.3. Nakonec se zde nachází hodnocení navrženého řešení oproti aktuální implementaci za použití různých měřítek, a to v podkapitole 5.1.

Pro návrh diagramů byl zvolen nástroj *Figma*. Tento nástroj umožňuje nejen vytvářet drátěné modely (v praxi se užívá spíše anglický výraz *wireframe*, proto bude nadále v této práci využíván), ale také různé diagramy dle potřeby, takže vyhovuje požadavkům pro návrh uživatelského rozhraní.

3.1 Analýza uživatelských požadavků

Tato podkapitola shrnuje veškeré uživatelské požadavky nástroje *Combine*. V této podkapitole bylo čerpáno ze zdroje, [6], který se zabývá teorií ohledně specifikace požadavků. Uživatelský požadavek má více definic. Například:

> Podmínka nebo schopnost, nutná pro vyřešení problému uživatele nebo dosažení cíle.

Definice převzata z [6].

Uživatelské požadavky lze rozdělit do dvou skupin, a to *Funkční* a *Nefunkční*.

- **Funkční** požadavky jsou takové požadavky, které vyjadřují co přesně a co vše má daný program dělat. Také se dá říct, že je to v podstatě popis, jak má program reagovat na vstupy a jaké mají být odpovídající výstupy.

- **Nefunkční** požadavky jsou takové požadavky, které přímo nedefinují interakci uživatele s uživatelským rozhraním.

Funkční požadavky pro nové rozhraní

- **Definice nastavení systému pod testem** - Uživatelé musí mít možnost vyplnit nezbytné informace o *systému pod testem* a mít k němu přístup po celou dobu používání aplikace.
- **Vytváření šablon** - Uživatelé musí mít možnost vytvářet si vlastní šablony, které budou moci využívat v dalších případných operací s nástrojem. Tato možnost urychlí uživatelům práci, protože pokud by chtěli nástroj používat opakovaně a měli by testovat podobný nebo stejný systém, tak se mohou vyhnout opětovnému vytváření existujících nastavení.
- **Přidávání parametrů** – Pro správnou práci nástroje je třeba, aby uživatel zadal parametry, které se mají kombinovat. Zde je ale třeba přidat možnost přednastavených parametrů, které by si uživatel vybral a nemusel je zadávat manuálně.
- **Datové typy parametrů** - Parametry by měly mít možnost nabývat následujících datových typů:
  - Integer - Celé číslo
  - Float - Desetinné číslo
  - Boolean - Pravdivostní hodnota
  - Enum - Výčet různých hodnot
  - String - Textový řetězec
  - Date - Datum
  - Time - Čas
  - E-mail - E-mailová adresa
  - Telephone Number - Telefonní číslo
  - Id - Identifikátor
- **Výběr mezi režimy parametrů** - U parametrů by se měla nacházet možnost přepínat mezi režimy výběru. To jsou konkrétně režimy *Výběr pomocí kritérií* a *Výběr pomocí předdefinovaných dat*. *Výběr pomocí předdefinovaných dat* se nachází pouze u parametrů, u kterých tato varianta nepostrádá smysl a to jsou parametry datových typů String, Integer a E-mail. U ostatních datových typů je rozumnější tato data zadávat na základě zadaných kritérií.
- **Editace parametrů** - Již přidané parametry musí být možné editovat, aby je uživatel nemusel pro stejný výsledek vymazat a znovu přidat. Z hlediska uživatelského komfortu by bylo vhodné, kdyby šlo také měnit pořadí parametrů. Tato možnost zvýší přehlednost, pokud by uživatel preferoval jiné pořadí parametrů.
- **Odebírání parametrů** - Uživatelům musí být umožněno odebírat parametry v případě, že by si jejich přítomnost rozmysleli.
- **Omezení** - Uživatel musí mít možnost v aplikaci přidat omezení, aby se předešlo vygenerování takových kombinací, které z logického hlediska nedávají smysl.
- **Odebírání omezení** - Přidaná omezení musí být odstranitelná.

- **Tlačítko pro generování výsledků** - V aplikaci se musí na přehledném místě nacházet tlačítko, kterým se momentální nastavení testu odešle do nástroje *Combine*, které toto nastavení zpracuje a vygeneruje odpovídající výsledek. Toto tlačítko musí být viditelné po celou dobu používání nástroje, aby mohl uživatel vygenerovat výsledek ihned po provedených změnách.
- **Zobrazení výsledků** - Po vygenerování výsledků se musí přehlednou formou vygenerovat tabulka, která bude představovat zkombinované parametry.
- **Export výsledků** - Po vygenerování výsledků se v aplikaci také musí nacházet možnost si v různých adekvátních formátech stáhnout tuto testovací sadu za účelem použití ve svých testech.
- **Export a Import nastavení systému pod testem** - Uživatel musí mít možnost vyexportovat si nastavení celého systému pod testem, které by pak mohl importovat a používat ho znovu tak, že může načtené nastavení upravit a vygenerovat nové výsledky.
- **Přerušení generování výsledků** - Z důvodu, že generování výsledku pro větší počet zadaných parametrů a jejich hodnot může být výpočetně náročné a dlouho trvající, bylo by vhodné, aby měl uživatel možnost generování přerušit.

Nefunkční požadavky pro nové rozhraní

- **Uživatelská přívětivost** - Aplikace musí být k uživatelům vstřícná a intuitivní. Musí se snadno ovládat a to i bez předchozí zkušenosti s používáním ať už tohoto konkrétního nástroje, tak i všeobecně testovacích nástrojů. Pokud uživatel zadá nějaký chybný vstup, musí být na tuto skutečnost upozorněn.
- **Paralelní přístup** - Aplikace musí být schopna zpracovávat požadavky od více uživatelů ve stejném čase a to zcela nezávisle na sobě.
- **Podpora prohlížečů** - Aplikace musí být kompatibilní s novými verzemi nejčastějších prohlížečů a to jmenovitě *Google Chrome*, *Mozilla Firefox* a *Microsoft Edge*.

3.2 Nedostatky aktuálního rozhraní

V první řadě je potřeba nedostatky rozdělit do kategorií. První kategorie obsahuje nedostatky na straně uživatelského rozhraní a druhá kategorie obsahuje nedostatky aplikačního rozhraní.

Nedostatky uživatelského rozhraní

Zde se nachází shrnutí nedostatků na straně uživatelského rozhraní, tedy takových nedostatků, které přímo omezují interakci uživatele s uživatelským rozhraním. U všech nedostatků je specifikováno, které principy použitelnosti uživatelských rozhraní nesplňují. O principech použitelnosti pojednává kapitola 2.1.1. Tyto nedostatky jsou nežádoucí věcí, protože snižují uživatelskou zkušenost, o které je zmínka v kapitole 2.1.

- N01 - Nevhodné využívání prostoru na obrazovce. Jednotlivé části rozhraní jsou příliš velké a nutí uživatele zbytečně posouvat vertikálně obrazovku pro zobrazení ostatních

částí aplikace. Lze si toho všimnout v příloze B.1, kde jsou jednotlivé komponenty daleko od sebe. Tento nedostatek porušuje princip použitelnosti 2.1.1, "Estetický a minimalistický design".

- N02 - Uživatel musí hodnoty parametrů psát sám. To není vhodné, protože uživatel ve většině případů nepotřebuje své konkrétní hodnoty, ale takové hodnoty, které by splňovaly jeho požadavky pro testovaný artefakt. Tento nedostatek nesplňuje princip použitelnosti 2.1.1, "Flexibilní a efektivní použití".

- N03 - Nemožnost editovat přidané parametry. Pokud by chtěl uživatel změnit nastavení u parametru, nezbývalo by mu nic jiného, než parametr odebrat a znovu jej přidat s požadovaným nastavením. V takovém případě uživatel ztratí čas a jeho uživatelská zkušenost by nebyla pozitivní. Tento nedostatek nesplňuje princip použitelnosti 2.1.1, "Uživatelská kontrola a svoboda".

- N04 - Za předpokladu, že by uživatel potřeboval testovat podobný nebo stejný systém pod testem, tak by musel veškeré nastavení, které už jednou musel provést, opakovat. Tento nedostatek nesplňuje princip použitelnosti 2.1.1, "Uživatelská kontrola a svoboda".

- N05 - Pokud uživatel zadá chybný vstup, není dostatečně informován o chybě, které se dopustil. Neví tak, který vstup je chybný a je potřeba jej změnit. Tento požadavek nesplňuje princip použití 2.1.1, "Pomoc uživatelům s chybami.

- N06 - Tlačítka pro přidávání parametrů, bloků a omezení nejsou stejná. Toto se nemusí nutně považovat za chybu, ale z hlediska konzistence by bylo lepší, kdyby všechna tato tlačítka byla indikována ikonami nebo textem. Tento nedostatek nesplňuje princip použití 2.1.1, "Konzistence a standardizace".

Nedostatky aplikačního rozhraní

Zde se nachází shrnutí nedostatků na straně aplikačního rozhraní, tedy takových nedostatků, nad kterými uživatel nemá kontrolu a odehrávají se na mezivrstvě mezi uživatelským rozhraním a serverovou částí.

- N07 - Nevhodná práce s JavaScript Object Notation (dále jen JSON) pro Representational State Transfer (dále jen REST) Application Programming Interface (Dále jen API). V aktuální implementaci se s každým atributem pracuje jako s řetězcem znaků. To je ale nesprávný přístup vzhledem k tomu, že se v této struktuře JSON nacházejí takové parametry, které lze reprezentovat jiným, korektním, datovým typem.

- N08 - Pokud uživatel zadal pro vygenerování úlohu s mnoha parametry a mnoha hodnotami parametrů, tak se generování výsledků stává časově delším. Chybí zde možnost pro uživatele, aby generování přerušil a mohl pokračovat v práci se stávajícím nastavením SUT.

3.3 Analýza řešení nedostatků

V této podkapitole se nacházejí řešení nedostatků, které byly shrnuty v předchozí podkapitole 3.2. Pro některá uvedená řešení jsou uvedeny klady a zápory.

Řešení nedostatků uživatelského rozhraní

V této podkapitole se nacházejí řešení nedostatků uživatelského rozhraní. Pro lepší pochopení těchto podkapitol se v příloze C nachází stavový diagram uživatelského rozhraní.

R01 - Nevhodné využití prostoru

Toto řešení se zabývá nedostatkem N01 3.2. Pro vyřešení tohoto problému byl vytvořen wireframe, jehož účelem je nastínit představu o rozložení jednotlivých prvků uživatelského rozhraní na obrazovce. Tento wireframe se nachází v příloze C. Daný wireframe by se dal rozdělit na několik částí.

- **Oddíl pro hlavní nastavení** - Nachází se nahoře vlevo a jeho účelem je mít přístup k často používaným akcím po celou dobu používání aplikace. Proto by se tento panel měl pohybovat společně s obrazovkou, když uživatel vertikálně pohybuje s obrazovkou v prohlížeči. Jmenovitě se v hlavním nastavení nachází postupně tyto možnosti:
  - Textové pole pro zadání názvu testovaného systému.
  - Rozbalovací seznam, ve kterém lze vybrat násobnost kombinací od čísla 1 do čísla 6.
  - Přepínací tlačítko pro výběr, zda mají být hodnoty označené jako ”don’t care” náhodné.
  - Přepínací tlačítko pro výběr, zda uživatel chce ve výsledné tabulce konkrétní hodnoty, nebo jejich identifikátory.
  - Tlačítko pro exportování aktuálního nastavení SUT.
  - Panel pro nahrání požadovaného nastavení SUT. Tento panel funguje jak způsobem, kdy se po kliknutí otevře prohlížeč souborů pro výběr nastavení, tak způsobem "táhni a pusť" (anglicky drag and drop).
  - Tlačítko pro vygenerování výsledku.
- **Oddíl pro zadávání parametrů** - Nachází se nahoře napravo a účelem této části je poskytnout uživateli možnost vkládání parametrů. Je zde připraveno tlačítko pro přidání řádků a číselný vstup, který reprezentuje kolik řádků by se mělo přidat. Bude se zde tedy manipulovat s editovatelnými řádky, které představují jednotlivé parametry. V návrhu jsou u parametrů řádky uvedeny obecně, protože každý datový typ vyžaduje jiná vstupní pole. Proto o blocích parametrů pojednává další podkapitola 3.3 - "Hodnoty u parametrů". Mohla by také vzniknout otázka, zdali by nebylo žádoucí generovat výsledek po každé úpravě nebo přidání parametru. Tato operace je však výpočetně náročná, a proto v tomto případě nevhodná.
- **Oddíl pro zadávání omezení** - Nachází se hned pod oddílem s parametry. Tento oddíl slouží k zadávání omezení pro eliminaci nesmyslných kombinací. Je zde možnost přidávat řádky pro omezení stejným způsobem jako v předchozím oddíle. Tyto řádky jsou pak postupně číslovány a u každého řádku je ikonka křížku, pomocí které lze řádek pro omezení odebrat.
- **Oddíl pro výsledné hodnoty** - Nachází se v dolní části stránky a je zobrazen až v momentě, kdy uživatel úspěšně provede akci pro vygenerování výsledků. Výsledky by měla reprezentovat tabulka, ve které budou uvedeny všechny výsledné kombinace

parametrů. Aby mohl uživatel tyto výsledky použít, nacházejí se zde také tlačítka pro stažení souborů s výsledky ve třech formátech, a to JSON, XML a CSV. Nabízí se také varianta v implementační fázi přidat varianty pro více formátů.

R02 - Hodnoty u parametrů

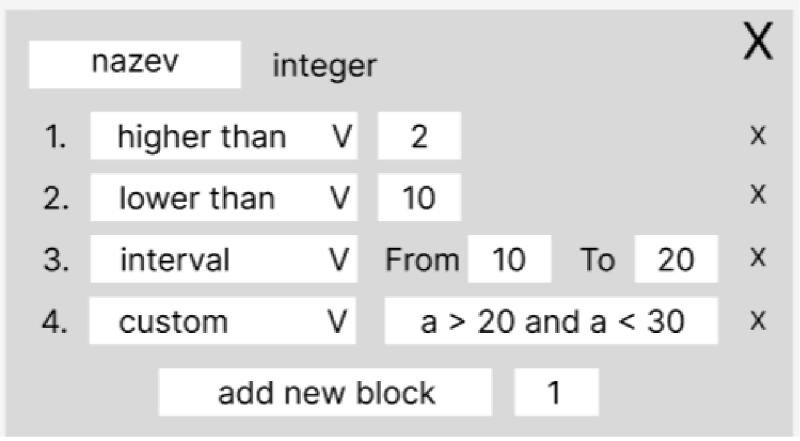
Toto řešení se zabývá nedostatkem N02 3.2. V aktuální implementaci musí uživatel vyplňovat jednotlivé bloky parametrů manuálně. Tento problém však lze vyřešit tak, že by byl uživateli nabídnut výběr pomocí dvou režimů. V prvním by uživatel vybíral z předdefinovaných datových sad a až pokud by ani zde uživatel nenalezl požadovanou volbu, tak by použil své vlastní konkrétní data, která by vložil manuálně. Tento režim má však svůj význam pouze u některých datových typů. Druhou volbou je zadání pomocí kritérií, kde by uživatel místo manuálního vypsání výrazu vybral požadovanou variantu kritéria (například interval u datového typu "integer").

Tento problém nelze řešit generickými bloky pro parametry, protože každý datový typ vyžaduje jiná vstupní pole pro zadávání bloků. Každý parametr kromě takových, kde je datovým typem "boolean" bude mít možnost výběru, kde uživatel může zadat vlastní výraz stejným způsobem, kterým se hodnoty zadávají v aktuální implementaci. Zde je podoba bloků parametrů pro každý datový typ, se kterým nástroj pracuje:

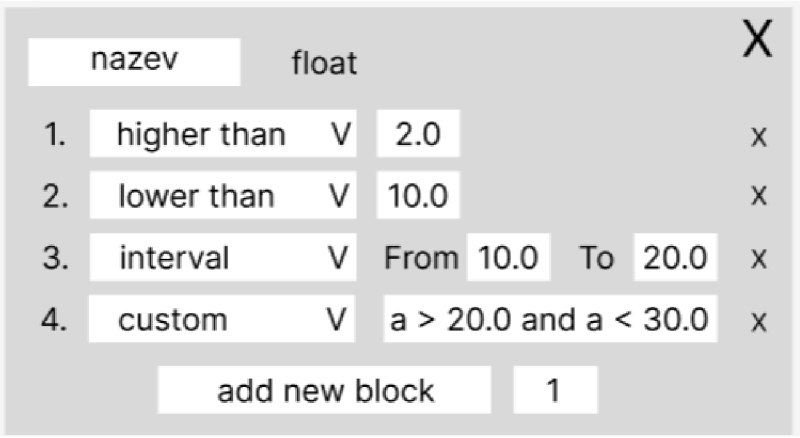
- **Parametr typu ”integer”** - Parametr typu "integer" nabízí uživateli zjednodušení v podobě předdefinovaných hodnot, kde možnosti jsou následující:
  - **Sudá čísla** - Výběr požadovaného počtu sudých čísel
  - **Lichá čísla** - Výběr požadovaného počtu lichých čísel
  - **Prvočísla** - Výběr požadovaného počtu prvočísel

  A u výběru pomocí kritérií jsou možnosti následující:
  - **Menší než** - Slouží pro zadání hodnoty, která značí taková celá čísla, která jsou menší než zadané celé číslo.
  - **Větší než** - Slouží pro zadání hodnoty, která značí taková celá čísla, která jsou větší než zadané celé číslo.
  - **Interval** - Slouží pro zadání hodnoty, která značí interval mezi dvěma celými čísly.
  - **Vlastní**

- **Parametr typu ”float”** - Tento typ parametru se možnostmi výběru neliší od parametru typu "integer", který je popsán výše. Jediným rozdílem je, že zde uživatel zadává desetinná čísla. Není zde možnost výběru z předdefinovaných hodnot.
  - **Menší než** - Slouží pro zadání hodnoty, která značí taková desetinná čísla, která jsou menší než zadané desetinné číslo.
  - **Větší než** - Slouží pro zadání hodnoty, která značí taková desetinná čísla, která jsou větší než zadané desetinné číslo.
  - **Interval** - Slouží pro zadání hodnoty, která značí interval mezi dvěma desetinnými čísly.
  - **Vlastní**

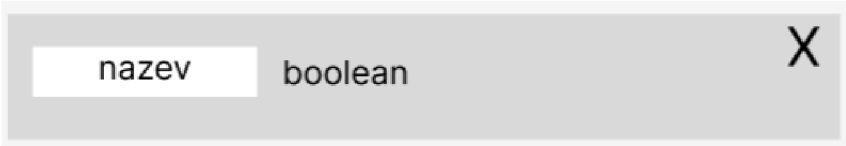


Obrázek 3.1: Ukázka parametru typu ”integer”



Obrázek 3.2: Ukázka parametru typu ”float”

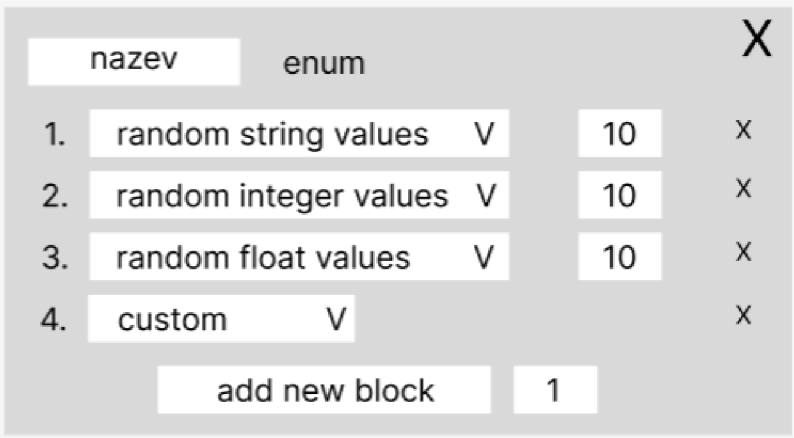
- **Parametr typu ”boolean”** - Tento typ parametru je nejprostější ze všech, neboť datový typ "boolean" může nabývat pouze parametrů *true* a *false*, a tak z tohoto důvodu nemá smysl jakýkoliv výběr hodnot pomocí kritérií, ani pomocí datových sad.



Obrázek 3.3: Ukázka parametru typu ”boolean”

- **Parametr typu ”enum”** - Parametr tohoto typu slouží k reprezentaci výčtu hodnot, které mohou nabývat různých datových typů. Proto zde lze uživateli ulehčit práci tak, aby mohl zadat, které datové typy by se měly ve výčtu objevit, a kolik by jich zde mělo být. Není zde možnost výběru pomocí kritérií.

- **Počet náhodných textových řetězců** - Vloží n náhodně vygenerovaných řetězců.
- **Počet náhodných celých čísel** - Vloží n náhodných čísel
- **Počet náhodných desetinných čísel** - Vloží n náhodných desetinných čísel
- **Vlastní**

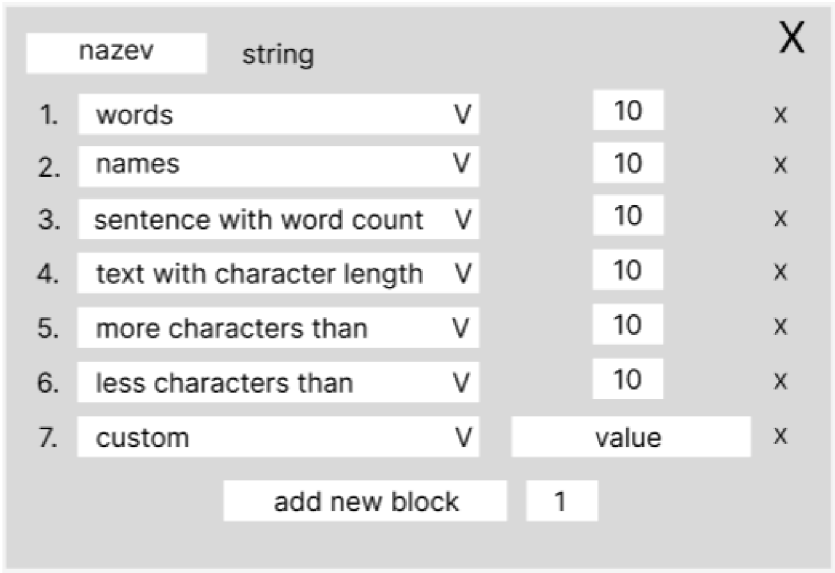


Obrázek 3.4: Ukázka parametru typu ”enum”

- **Parametr typu ”string”** - Parametr typu "string" nabízí zjednodušení v podobě výběru uživatelem určeného počtu řetězců, které pak budou náhodně vloženy z konfiguračního souboru testovacích dat. Dále tento datový typ nabízí uživateli vygenerování náhodných řetězců v zadaném rozmezí znaků. Výběr z předdefinovaných hodnot je následující:
  - **Slova** - Vloží n náhodně vybraných slov z konfiguračního souboru
  - **Jména** - Vloží n náhodně vybraných jmen z konfiguračního souboru
  - **Měsíce** - Vloží názvy všech měsíců v roce
  - **Věta s počtem slov** - Vloží text s n slovy

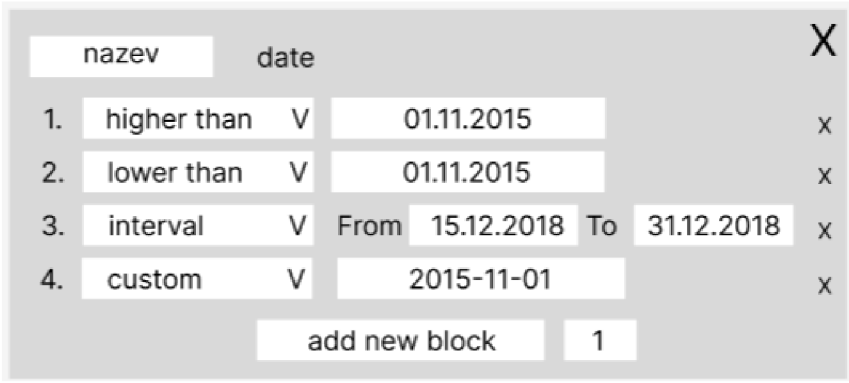
  U výběru pomocí zadání kritérií jsou možnosti následující:
  - **Text s počtem znaků** - Text musí mít zadaný počet znaků
  - **Text s větším počtem znaků než** - Text musí mít větší počet znaků než počet, který je zadán
  - **Text s menším počtem znaků než** - Text musí mít menší počet znaků než počet, který je zadán
  - **Vlastní**

- **Parametr typu ”date”** - U bloků parametrů typu "date" jsou následující možnosti zadávání hodnot, kde k výběru hodnot bude sloužit speciální komponenta pro výběr data. Není zde možnost výběru z předdefinovaných hodnot.
  - **Menší než** - Slouží pro zadání hodnoty, která značí taková data, která jsou menší než zadané datum.



Obrázek 3.5: Ukázka parametru typu ”string”

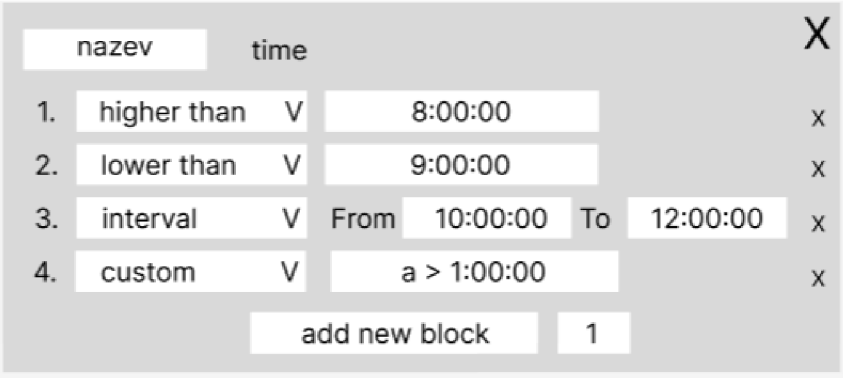
- **Větší než** - Slouží pro zadání hodnoty, která značí taková data, která jsou větší než zadané datum.
- **Interval** - Slouží pro zadání hodnoty, která značí interval mezi dvěma zadanými daty.
- **Vlastní**



Obrázek 3.6: Ukázka parametru typu ”date”

- **Parametr typu ”time”** - Parametry tohoto datového typu uživateli nabízejí stejný výběr možností, jako parametr datového typu "date". Místo data se zde však vybírá čas, a to pomocí speciálního pole pro výběr času. Není zde možnost výběru z předdefinovaných hodnot.
  - **Menší než** - Slouží pro zadání hodnoty, která značí takové časy, které jsou menší než zadaný čas.
  - **Větší než** - Slouží pro zadání hodnoty, která značí taková časy, které jsou větší než zadaný čas.

– **Interval** - Slouží pro zadání hodnoty, která značí interval mezi dvěma zadanými časy.
– **Vlastní**



Obrázek 3.7: Ukázka parametru typu ”time”

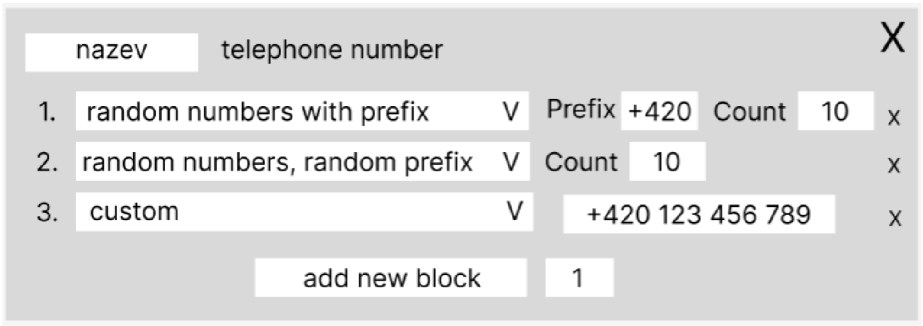
- **Parametr typu ”e-mail”** - Parametry tohoto datového typu uživateli nabízejí výběr z následujícího typu předdefinovaných dat:
  – **Náhodné e-mailové adresy** - Vloží uživatelem zadaný počet náhodně vybraných e-mailových adres z konfiguračního souboru.
  – **Vlastní**

  U výběru pomocí kritérií jsou možnosti následující:
  – **Vygenerovaná e-mailová adresa s délkou** - E-mailová adresa musí mít délku o zadaném počtu znaků.
  – **Vygenerovaná e-mailová adresa s délkou jména a domény** - E-mailová adresa musí mít délku jmenné částí e-mailové adresy o zadané délce a délku doménové částí o zadané délce.
  – **Vlastní**

Obrázek 3.8: Ukázka parametru typu ”e-mail”

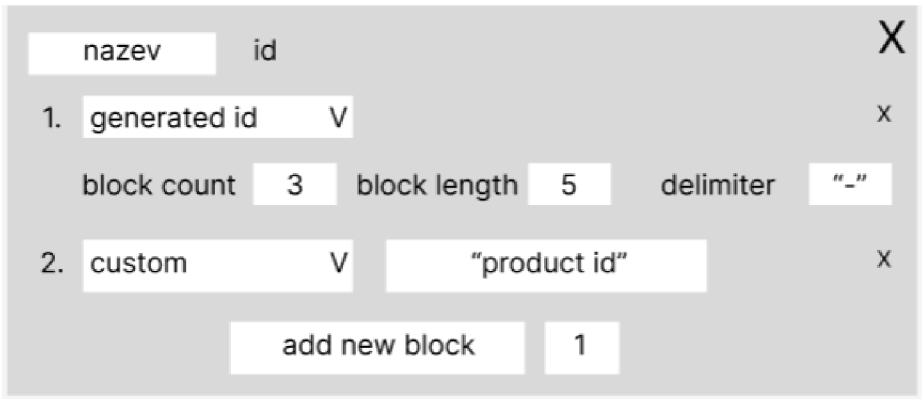
- **Parametr typu ”telephone number”** - Parametry tohoto typu nabízejí následující možnosti výběru, kde není možnost výběru z předdefinovaných dat.

- **Zadaným prefix** - Telefonní číslo musí obsahovat zadaný prefix.
- **Delší než** Telefonní číslo musí být delší než zadaný počet.
- **Kratší než** Telefonní číslo musí být kratší než zadaný počet.
- **Vlastní**



Obrázek 3.9: Ukázka parametru typu ”telephone number”

- **Parametr typu ”id”** - Parametry tohoto typu uživateli nabízejí výběr následujících možností, kde není možnost výběru z předdefinovaných dat.
  - **Vygenerované ID** - Vloží náhodně vygenerované ID skládající se z alfanumerických znaků a oddělovače se zadaným počtem bloků, zadanou délkou jednoho bloku a zadaným oddělovačem mezi jednotlivými bloky. Tedy například "abc-def-123".
  - **Vlastní**



Obrázek 3.10: Ukázka parametru typu ”id”

R03 - Chybějící editace parametrů

Toto řešení se zabývá nedostatkem N03 3.2. Jak je uvedeno v předchozí podkapitole, pro každý datový typ bude mít parametr vlastní komponentu, která bude mít různá vstupní pole. Všechny druhy komponent ale budou editovatelné ve smyslu změny obsahu dat. Druhý způsob editace, který by měl také být v nové implementaci je změna pořadí parametrů. Ta by měla probíhat za pomocí metody "táhni a pusť" (anglicky drag and drop).

R04 - Nelze opakovaně použít nastavení

Toto řešení se zabývá nedostatkem N04 3.2. Tento problém je v návrhu řešen tak, že se v panelu s hlavním nastavením nacházejí 2 tlačítka. První je pro stažení aktuálního nastavení v souboru, pro který bude využit formát JSON. Druhé tlačítko v návrhu má sloužit k nahrání nastavení. Pokud uživatel provede akci pro nahrání souboru s nastavením, objeví se modální okno (uvedeno v příloze C, ve kterém lze soubor vybrat buď prohlížením v adresáři, nebo metodou "táhni a pusť" (anglicky drag and drop). Modální okno se po vybraném souboru automaticky uzavře, protože uživateli je nabídnuto vybrat pouze jeden soubor a nemusí pak provést kliknutí navíc kvůli potvrzení výběru.

R05 - Zpětná vazba o chybách

Toto řešení se zabývá nedostatkem N05 3.2. Tento problém lze vyřešit tak, že se do uživatelského rozhraní přidá prostor, který bude určen k vyobrazení chybových stavů. Ideální pozice je v pravém dolním rohu. Zde se po každé chybné operaci zobrazí zpráva, která bude červenou barvou indikovat chybu a bude zde srozumitelná chybová hláška.

R06 - Nekonzistence prvků

Toto řešení se zabývá nedostatkem N06 3.2. Nedostatek lze vyřešit poměrně jednoduše, a to tak, že se vzhled všech tlačítek pro přidávání sjednotí do ikony symbolu plus, který indikuje tlačítko pro přidání prvku.

Řešení nedostatků aplikačního rozhraní

R07 - Nevhodná práce s JSON

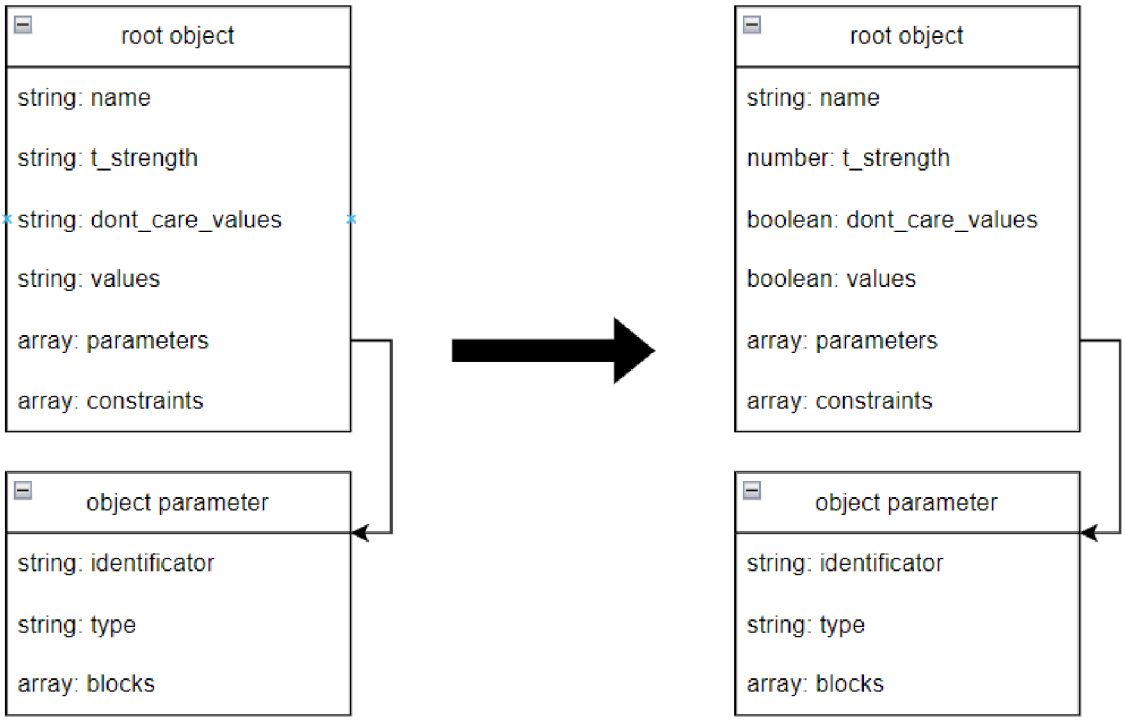
Toto řešení se zabývá nedostatkem N07 3.2. Tento nedostatek lze vyřešit změnou nevhodně přidělených datových typů na takové, které jsou pro přenášená data vhodnější. Pro ukázku, jak by tato změna datových typů měla vypadat, je zde přiložen obrázek 3.11:

R08 - Chybějící možnost přerušení generování výsledku

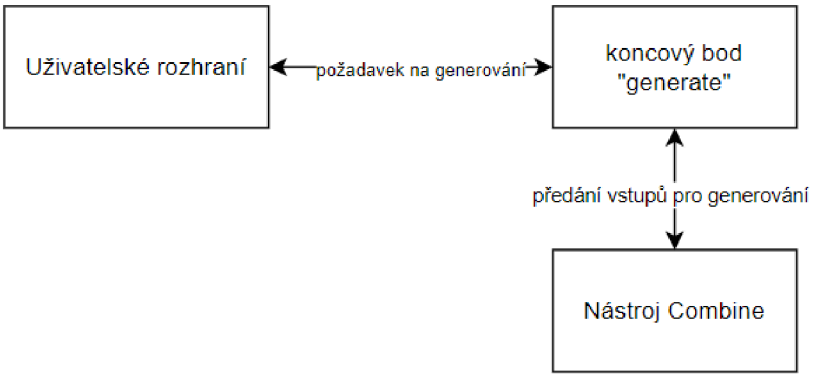
Toto řešení se zabývá nedostatkem N08 3.2. Vzhledem k tomu, že proces kombinování vstupních parametrů je mnohdy časově náročný, bylo by vhodné uživateli vizuálně prezentovat, že se nachází ve stavu generování výsledků. Dále by vedle zobrazení tohoto stavu mělo být tlačítko, které umožní uživateli proces generování ukončit.

Aktuální aplikační rozhraní se skládá pouze z jednoho koncového bodu *generate*, která se stará o zpracování vstupů, které jsou zde předány z uživatelského rozhraní. S těmito vstupy se provede právě kombinace parametrů a výsledek kombinace je předán zpět na uživatelské rozhraní, kde je výsledek prezentován uživateli.

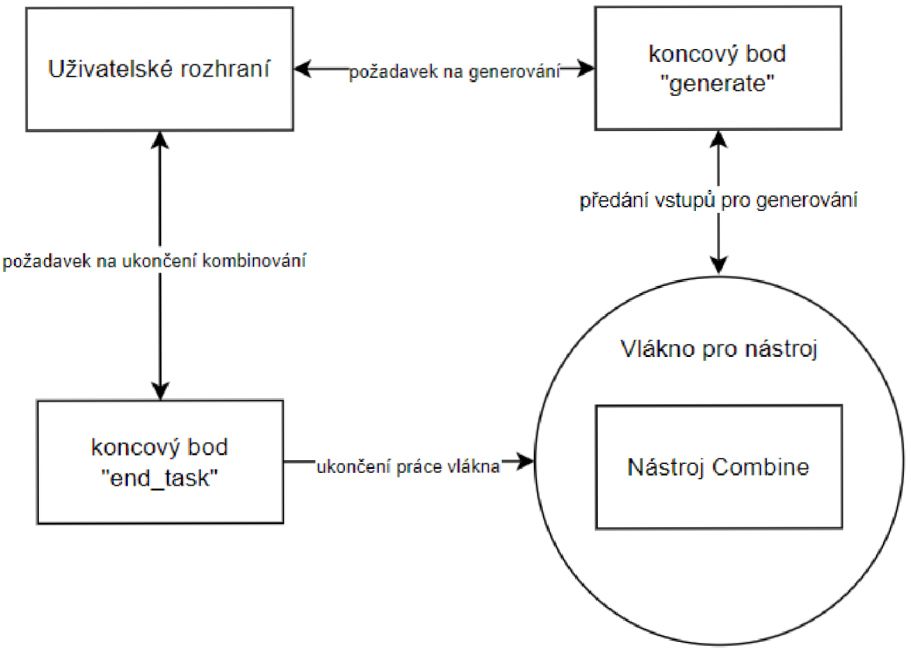
Cílem je tedy vytvořit nový koncový bod, který se postará právě o přerušení aktuálního procesu kombinování. Vybraným způsobem je využití více vláken. Konkrétně by aplikační rozhraní mělo obsahovat dva koncové body, kde koncový bod "generate" bude změněn tak, aby při požadavku vytvořil nové vlákno, na kterém bude tato úloha probíhat. Nově navržený koncový bod "end_task" v případě příchozího požadavku probíhající vlákno pro kombinování parametrů ukončí a předá informaci o úspěšném ukončení úlohy zpět uživatelskému rozhraní.



Obrázek 3.11: Zobrazení změny datových typů dokumentu JSON



Obrázek 3.12: Blokové schéma aktuálního aplikačního rozhraní



Obrázek 3.13: Blokové schéma navrženého aplikačního rozhraní

Kapitola 4

Implementace rozhraní

Tato kapitola se zabývá implementační fází této práce. Cílem implementace je vytvořit funkční a uživatelsky přívětivé uživatelské rozhraní ve webovém prohlížeči, který umožňuje práci s nástrojem Combine. Jako první v podkapitole 4.1 ukazuje výběr nástroje a nachází se zde také jeho popis. V další podkapitole 4.2 je popsána struktura aplikace, a jak jsou jednotlivé části propojeny. A nakonec v podkapitole 4.3 jsou ukázány implementační detaily jednotlivých komponent. Nově implementované uživatelské rozhraní lze vidět v příloze E.1.

4.1 Technologie a nástroje

Tato podkapitola se zabývá vybranými technologiemi a nástroji, které byly využity k tvorbě aplikace. Jedná se především o webové technologie zaměřené na tvorbu klientské části webové aplikace, ale některé jsou zaměřeny i na druhou, serverovou část webové aplikace.

4.1.1 React

Vybraným nástrojem pro implementaci uživatelského rozhraní se stala knihovna jazyka JavaScript, *React*. Tato knihovna umožňuje tvorbu uživatelských rozhraní pro webové aplikace a mezi její hlavní výhody patří jednoduchost a výkonnost. Jedním z nejvíce oceňovaných principů knihovny React je komponentová architektura. To znamená, že celou stránku lze rozdělit do jednotlivých komponent, které lze znovu využívat, což zvyšuje znovupoužitelnost a generičnost komponent, respektive celého kódu. Dalším principem knihovny *React* je Virtual Document Object Model (Virtual DOM). Pomocí této techniky se minimalizuje používání skutečného DOM, což zvyšuje výkon webové aplikace. Mezi další moderní alternativy knihovny *React* patří například *Angular* nebo *Vue.js*. Oba dva tyto frameworky jsou taktéž pro skriptovací jazyk JavaScript.

4.1.2 Material UI

Material UI je knihovna, která je součástí balíčku MUI System, což je sada CSS balíčku, které vytvářejí již hotové komponenty pro nastylování stránky. Z tohoto balíčku nové uživatelské rozhraní využívá především vstupní komponenty, ale i takové, které se starají o rozložení prvků na stránce. V této sekci byly informace získány z [5].

4.1.3 REST API

Tato sekce využívá informací ze zdroje [4]. REST API (anglicky The Representational State Transfer Application Programming Interface) je konkrétní druh API, zatímco REST je architektonický styl, který obsahuje sadu pravidel a zásad, kterými se takové API musí řídit. Webové služby, které splňují architekturu REST jsou nazývány jako RESTful webové služby. Mezi hlavní principy REST patří:

- Každý zdroj musí být unikátně identifikován a musí být adresovatelný pomocí URI.
- Klientská strana aplikace nesmí znát vnitřní reprezentaci dat, ale pracuje s vnější reprezentací dat jako například *XML* nebo *JSON*.
- Ke zdrojům API je přistupováno pomocí standardních metod HTTP protokolu jako třeba Post, Get, Put nebo Delete. Každá metoda má definované jiné chování pro daný zdroj.
- Komunikace mezi klientem a serverem je bezstavová. To znamená, že každý požadavek obsahuje všechny potřebné informace a neudržuje se žádný stav interakce na straně serveru.

REST API tedy znamená takové API, které splňuje všechna pravidla a zásady pro komunikaci REST architektury.

4.1.4 Flask

Flask je webový framework pro vytváření webových aplikací, který se stará o serverovou část aplikace. V tomto frameworku lze vytvářet API (anglicky Application Programming Interface), které se skládá z takzvaných koncových bodů (anglicky Endpoints), což jsou vlastně URL adresy definované na straně serveru. O konkrétní verzi API, REST API pojednává sekce 4.1.3. Klient pak dokáže se serverem komunikovat pomocí HTTP požadavků. U těchto koncových bodů je v knihovně Flask implementována obslužná funkce, která se při požadavku zaslaném na koncový bod postará o zpracování požadavku a odpovědi na tento požadavek zpět ke klientovi.

4.2 Struktura aplikace

Tato sekce popisuje strukturu aplikace. V sekci 4.2.2 je popsána práce s daty na úrovni uživatelského rozhraní, v sekci 4.2.3 je popsáno, jakým způsobem se získávají náhodné hodnoty z datových sad pro definice bloků parametrů a v sekci 4.2.4 je popsáno, jak funguje propojení klientské a serverové části aplikace.

Klientská část aplikace, která je vytvořena v knihovně React, se dělí do komponent. Kořenem celé této struktury je komponenta *App.js*, která se pak dále rozděluje do čtyř větví, kde každá implementuje jinou část aplikace. Toto rozdělení lze vidět v obrázku 4.1. Tyto části by se daly rozdělit následovně:

- Parametry
- Testovací nastavení
- Tabulka s výsledky

- Omezení

Jednotlivé komponenty z těchto kategorií jsou popsány v sekci 4.3. Každá ze zmíněných částí rozvětvení má v projektu svůj adresář pro komponenty spadající do dané větve. Tyto části aplikace jsou nastylovány a umístěny tak, aby byly podle řešení nedostatku R01 3.3.

4.2.1 Adresářová struktura zdrojového kódu

V této sekci se nachází seznam hlavních částí adresářové struktury aplikace, kde se nachází převážně zdrojový kód. Tučně jsou vyznačeny složky a tence jsou vyznačeny soubory. Z důvodu velkého množství souborů v klientské část ve složce "src" jsou zde uvedeny pouze složky. U jednotlivých sekcí je pak uveden název souboru, ve kterém je daná část implementována a ve které ze složek daný soubor hledat.

- **backend**
  - api.py
  - **DataSets**
    - Primes.txt
    - Words.txt
    - ... a tak dále
- **src**
  - **BlockLineComponents**
    - **Date**
    - **Number**
    - **String**
    - **Time**
  - **ConstraintComponents**
  - **Contexts**
  - **Data**
  - **ParameterComponents**
    - **ParameterContentsComponents**
  - **ResultComponents**
  - **Tests**
  - **TestSettingsComponents**

4.2.2 Práce s daty

Práce s daty v aplikaci funguje pomocí jedné z funkcionalit knihovny React, a to Context API. Tato technika v podstatě umožňuje vytvořit datovou strukturu, která by měla být globální a lze ji číst a editovat. Lze tím říct, že kontext je pro aplikaci globálním úložištěm dat, ke kterým lze mít přístup z celé aplikace i bez závislostí komponent. Nevýhodou tohoto přístupu je však horší testovatelnost. Mají v projektu vytvořen svůj adresář s názvem *Contexts*.

Konkrétně jsou implementovány dva kontexty a oba se nachází ve složce *Contexts*. *SutSettingsContext* slouží k uchování dat všech vstupů, které uživatel zadává. Udržuje tedy hodnoty nastavení SUT, omezení, parametrů a jejich bloků. Kontext *ResultContext* slou

4.2.3 Datové sady

Datové sady fungují tak, že je v klientské částí vytvořen soubor formátu JSON s názvem *parameters.json*, který specifikuje a přiřazuje jednotlivé datové sady k datovým typům. Například pro datový typ *Integer* jsou zde datové sady *Primes, Odd a Even*. Z tohoto souboru se vybírají typy specifikace bloků do rozbalovacího seznamu pro jednotlivé parametry, kde v tomto seznamu lze pak vybrat požadovaný typ výběru. Tato část řeší nedostatek R02 3.3.

Pokud uživatel přidá typ bloku, který je určen v souboru *parameters.json* jako datová sada daného parametru, tak se konkrétní hodnoty z datové sady získají pomocí HTTP požadavku, což je popsáno v sekci 4.2.4. Tyto datové sady jsou totiž uloženy v jednotlivých souborech ve složce "DataSets" na serverové straně. Soubory jsou pojmenovány stejně jako v souboru *parameters.json*. API poté rozhodne, ze kterého souboru má hodnotu získat. Z vybraného souboru pak náhodně vybere hodnotu a vrátí ji zpět ke klientovi za pomocí HTTP odpovědi. Zároveň se kontroluje, zda se tato hodnota již v některém bloku parametrů nachází. Pokud tam již je, tak se vybere jiná hodnota a vrátí se ta. Za předpokladu, že by v datové sadě došly hodnoty, které se ještě v blocích parametrů nenachází, tak dojde k chybě, která je oznámena prostřednictvím chybové hlášky.

4.2.4 Komunikace s API

Komunikace s API probíhá za pomocí REST API, které je popsáno v sekci 4.1.3. Zde je implementován jediný koncový bod (anglicky endpoint), který se stará o získávání náhodných hodnot z vybraných datových sad, které jsou uloženy na straně serveru. Tento endpoint je implementován v souboru "api.py", který se nachází ve složce "backend".

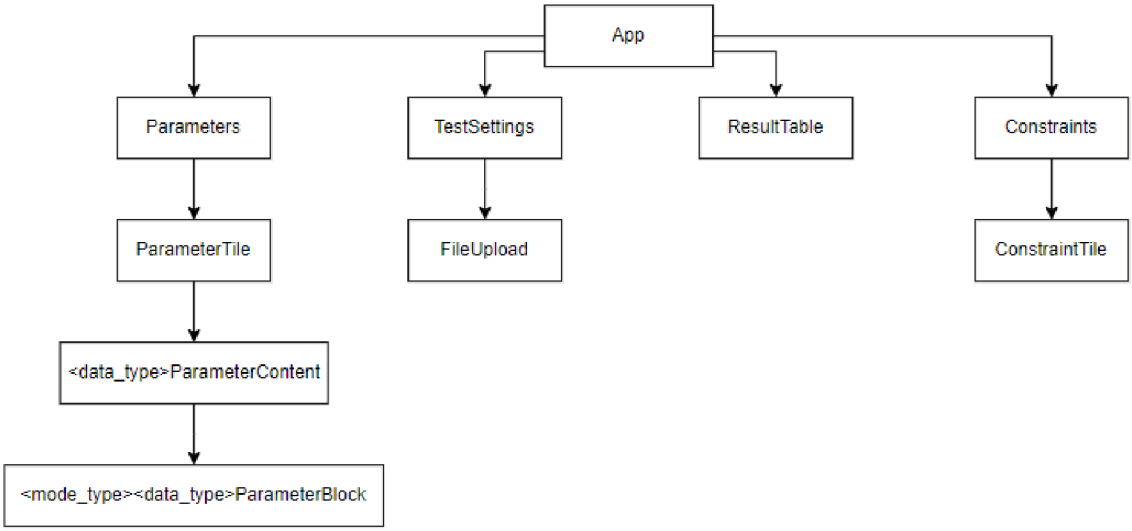
Datové sady zde lze manuálně přidávat a odebírat, ale je taktéž potřeba v souboru *parameters.json* přiřadit tuto novou datovou sadu ke konkrétnímu datovému typu. Jak funguje JSON soubor popisující vztahy mezi datovým typem a datovou sadou je popsáno v sekci 4.2.3.

Dalším využitým koncovým bodem, který však nespadá do tohoto projektu je nástroj Combine, který je potřeba nainstalovat společně s projektem. Do tohoto nástroje se po stisknutí tlačítko pro generování zašle JSON v požadované struktuře a a vrací JSON s výsledky. Komunikace s tímto koncovým bodem se nachází v komponentě 4.3.1.

V návrhu se nachází také řešení nedostatku R07 3.3 a R08 3.3. Tyto dvě řešení nakonec nebyly implementovány, protože nespadají do tohoto projektu a měly by být řešeny přímo při reimplementaci nástroje Combine.

4.3 Komponenty

Tato sekce se zabývá popisem jednotlivých komponent. Jak je napsáno v kapitole 4.1.1, knihovna React umožňuje vytváření komponent pro znovupoužitelnost v kódu. Jednotlivé komponenty se postupně zanořují do sebe, a vytváří tak stromovou strukturu. Komponenty mohou být vytvořeny jako třída nebo funkce, kde vrací JSX, což je vlastně HyperText Markup Language (HTML), upravené pro snadný zápis v kódu využívajícím React. Dále tyto komponenty obsahují veškerou přiřazenou logiku k této komponentě a nakonec důležitou součástí komponent jsou stavy. Ty umožňují komponentě uchovávat si svoje vnitřní data, se kterými se v komponentě pracuje. Jak jsou do sebe komponenty zanořeny lze vidět na obrázku 4.1. Každá komponenta má vlastní soubor a všechny tyto soubory se nacházejí, různě zanořené, ve složce *src*.



Obrázek 4.1: Schéma komponent

App

Tato komponenta je pro celou aplikaci vstupní a lze ji považovat za kořen celého stromu komponent. Elementy, které jsou potomky této komponenty jsou obaleny do kontextu, o kterém pojednává kapitola 4.2.2. Dále se pak dělí do čtyř větví, kde každá implementuje jinou část aplikace. O tomto rozdělení pojednává sekce 4.2. Komponenta se nachází v souboru *App.js* a tento soubor se nachází ve složce *src*.

Parameters

Tato komponenta se stará o zobrazení a přidávání parametrů. Obsahuje číselný vstup, který indikuje, kolik parametrů má být přidáno po stisku tlačítka. Dále pak mapuje data s parametry na dlaždice parametrů, o kterých pojednává sekce 4.3. Komponenta se nachází v souboru *Parameters.js*, který se nachází ve složce *ParameterComponents*.

Parameter Tile

Zde se nachází řešení nedostatku R03 3.3. Tato komponenta se stará o vykreslení dlaždice parametru. Obsahuje index, který určuje pořadí parametru v množině parametrů, textové pole pro identifikátor, který určuje označení parametrů ve výsledné testovací sadě a rozbalovací seznam, ve kterém se vybírá datový typ parametru.

Dále je v této komponentě skryta logika, která dle aktuálně zvoleného datového typu vybere, která z komponent pro obsah s bloky se má vykreslit. Tyto komponenty pro obsah s bloky jsou popsány v sekci 4.3. Pokud je zvolený datový typ "boolean", tak se obsah s bloky ani sekce s přidáváním bloků nevykreslí, protože zde jsou explicitně nastaveny 2 bloky "true" a "false".

Poslední část dlaždice parametru slouží k přidávání bloků a skládá se z číselného vstupu, který určuje kolik bloků bude přidáno. Uprostřed je tlačítko, kterým se bloky přidají a napravo od tlačítka je rozbalovací menu, kterým se vybírá režim přidání bloků. Tyto režimy jsou popsány v sekci 3.3 a v tabulce 4.1 lze vidět, které datové typy obsahují možnosti

přidání hodnot daným způsobem. Komponenta se nachází v souboru *ParameterTile.js*, který se nachází ve složce *ParameterComponents*.

Tabulka 4.1: Options for Data Types

Data Type	**Options**
Integer	Prvočísla, Lichá čísla, Sudá čísla, "Větší než" "Menší než", "Interval", Vlastní
String	Slova, Křestní jména, Měsíce "Více znaků než", "Méně znaků než", Vlastní
Float	"Větší než", "Menší než", "Interval", Vlastní
Enum	Řetězce, Celá čísla, Desetinná čísla, Vlastní
Date	"Větší než", "Menší než", "Interval", Vlastní
Time	"Větší než", "Menší než", "Interval", Vlastní
Boolean	-
Telephone Number	Vlastní
ID	Vlastní
Email	E-mailové adresy, Vlastní

Parameter Content

Tyto komponenty se starají o řešení nedostatku R02 3.3. Ve skutečnosti je těchto komponent více, kde pro každý datový typ kromě typu "boolean" se v adresáři *ParameterContents* nachází jeho vlastní komponenta pro obsah s bloky. Každá tato komponenta se stará o vykreslení adekvátní komponenty pro blok, o kterém pojednává sekce 4.3. Komponenty pro bloky jsou vybírány dle tabulky výše 4.1. Všechny komponenty pro obsah s bloky v následujícím seznamu se nacházejí ve složce *ParameterContentsComponents* :

- Date Parameter Content v souboru *DateParameterContent.js*
- Email Parameter Content v souboru *EmailParameterContent.js*
- Enum Parameter Content v souboru *EnumParameterContent.js*
- ID Parameter Content v souboru v souboru *IdParameterContent.js*
- Number Parameter Content v souboru *NumberParameterContent.js*
- String Parameter Content v souboru *StringParameterContent.js*
- Telephone Number Parameter Content v souboru *TelNumberParameterContent.js*
- Time Parameter Content v souboru *TimeParameterContent.js*

Block

Těchto komponent je ve skutečnosti také více. Nacházejí se v adresáři *BlockLineComponent*, kde se nacházejí podadresáře pro každý datový typ a v těchto podadresářích jsou komponenty implementující jednotlivé bloky. Popis těchto bloků se nachází v sekci 3.3. Speciálním

případem zde však je *CustomParameter*, který se stará o výchozí prázdnou definici, kterou lze zapsat pomocí staré gramatiky, která je popsána v manuálu nástroje Combine [8]. Je zde však změna, kdy je třeba textový popis bloků zapisovat do jednoduchých uvozovek místo dvojitých uvozovek. Na druhé straně je však tato komponenta také využita pro zápis dat z předdefinovaných datových sad. Pokud je této komponentě nastaven v argumentech příznak *useDataSet*, tak se pokusí podle nastaveného režimu bloku načíst ze serveru náhodně vybraný prvek z datové sady daného datového typu. Tento proces je popsán v sekci 4.2.3. Zde je seznam komponent bloků pro jednotlivé datové typy:

- **Date** - Tyto soubory se nacházejí ve složce *BlockLineComponents/Date*.
  - Higher Than Date Block v souboru *HigherThanDateBlock.js*
  - Lower Than Date Block v souboru *LowerThanDateBlock.js*
  - Interval Date Block v souboru *IntervalDateBlock.js*
- **Number** - Tyto soubory se nacházejí ve složce *BlockLineComponents/Number*.
  - Higher Than Number Block v souboru *HigherThanNumberBlock.js*
  - Lower Than Number Block v souboru *LowerThanNumberBlock.js*
  - Interval Number Block v souboru *IntervalNumberBlock.js*
- **Time** - Tyto soubory se nacházejí ve složce *BlockLineComponents/Time*.
  - Higher Than Number Block v souboru *HigherThanNumberBlock.js*
  - Lower Than Number Block v souboru *LowerThanNumberBlock.js*
  - Interval Number Block v souboru *IntervalNumberBlock.js*
- **String** - Tyto soubory se nacházejí ve složce *BlockLineComponents/String*.
  - Less Characters Than Block v souboru *LessCharacterThanBlock.js*
  - More Characters Than Block v souboru *MoreCharacterThanBlock.js*
- **Všechny datové typy** - Tento soubor se nachází ve složce *BlockLineComponents*.
  - Custom Block v souboru *CustomBlock.js*

Constraints

Tato komponenta se stará o zobrazení a přidávání omezení. Obsahuje číselný vstup, který indikuje, kolik omezení má být přidáno po stisku tlačítka. Dále pak mapuje data s omezeními na dlaždice omezení, o kterých pojednává sekce 4.3. Tato komponenta se nachází v souboru *Constraints.js*, který je ve složce *ConstraintsComponents*.

Constraint Tile

Tato komponenta se stará o vykreslení dlaždice pro omezení. Tato dlaždice obsahuje index, který značí pořadí v množině těchto omezení. Dále obsahuje textové pole, do kterého lze zapsat definici omezení podle gramatiky, která je popsána v manuálu nástroje Combine [8]. Konečně je v této dlaždici i tlačítko, které vymaže toto omezení. Tato komponenta se nachází v souboru *ConstraintTile.js*, který je ve složce *ConstraintsComponents*.

Result Table

Tato komponenta slouží k zobrazení výsledků po operaci generování, která je popsána v sekci 4.3.1. Výsledek je uspořádán do tabulky, kde jsou očíslovány jednotlivé řádky. V hlavičkách sloupců se nacházejí identifikátory parametrů, ke kterým daný sloupec patří. Při otevření webové aplikace je tato komponenta skryta a objeví se až po úspěšném vygenerování kombinací. Dále je skryta při načítání a je místo ní panel indikující načítání vygenerovaných kombinací. Také je zde nachystáno tlačítko, které bude sloužit k přerušení generování testovacích dat. Důvod proč není implementován se nachází zde 4.2.4. Tato komponenta se nachází v souboru *ResultTable.js*, který je ve složce *ResultComponents*.

4.3.1 Test Settings

Tato komponenta se stará o testovací nastavení. V první části obsahuje textový vstup pro zadání názvu systému pod testem, rozbalovací menu, pro určení T-strength (vysvětleno v sekci 2.2), přepínač pro výběr, zda mají být hodnoty randomizovány a přepínač pro výběr, zda chce uživatel do výsledku konkrétní hodnoty, nebo pouze index bloků.

Ve druhé části se nachází řešení nedostatku R04 3.3. Komponenta obsahuje vstup pro nahrávání souborů, jehož komponenta je popsána v sekci 4.3.1. Tento vstup pro soubory slouží k nahrávání dříve exportovaných nastavení, která lze znovu použít za účelem snížení potřebného času pro vyplnění nových testovacích nastavení. Druhé tlačítko slouží právě pro export aktuálně nastavených testovacích nastavení. Po kliknutí na toto tlačítko se stáhne soubor s názvem *sutSettings.json*. Pokud pak bude uživatel v budoucnu chtít tento vyexportovaný soubor použít, stačí ho vložit do vstupu pro nahrání těchto souborů. Poslední je tlačítko pro vygenerování testovací sady pomocí nástroje Combine. Tento proces je popsán v sekci 4.2.4. Tato komponenta se nachází v souboru *TestSettings.js*, který se nachází ve složce *TestSettingsComponents*.

File Upload

Tato komponenta slouží k nahrání souborů. Řeší požadavek P Soubor lze nahrát dvěma způsoby. První je, že po kliknutí na tuto komponentu se otevře prohlížeč souborů, ve kterém stačí vybrat požadovaný soubor. Druhý způsob je, že lze pomocí metody drag and drop přetáhnout požadovaný soubor přímo do tohoto vstupu. Po nahrání souboru se zkontroluje, zda je vložený soubor validní. Pokud je validní, tak se zobrazí hláška o úspěchu operace a tyto nahrané data z JSON souboru se uloží do kontextu, který je popsán v sekci 4.2.2. Pokud soubor není validní, tak je vypsána chybová hláška. Tím je řesen nedostatek R05 3.3 Tato komponenta se nachází v souboru *FileUpload.js*, který se nachází ve složce *TestSettingsComponents*.

Kapitola 5

Testování uživatelského rozhraní

Tato kapitola se zabývá otestováním návrhu a implementace nového uživatelského rozhraní. Sekce 5.1 se věnuje ověření, zda nově navržené uživatelské rozhraní vylepšuje stávající v podobě metrik počtu kliknutí a napsaných znaků. Sekce 5.3 se věnuje otestování implementovaného uživatelského rozhraní pomocí automatických testů grafického uživatelského rozhraní.

5.1 Metriky pro hodnocení uživatelského rozhraní

Tato kapitola má za cíl ukázat přínos předpokládaného vylepšení uživatelského rozhraní nástroje *Combine* oproti aktuální implementaci na konkrétních příkladech.

Počet interakcí

V této kategorii je ukázáno, kolik interakcí musí uživatel provést pro dosažení cíle definovaného zadaným scénářem. Jako interakce je v této podkapitole uvažováno kliknutí myší a počet napsaných znaků na klávesnici. Hodnotící scénáře musí být pro jasné pochopení definovány co nejpřesněji. Pro splnění požadovaných cílů by se pak měl průchod aplikací brát jako ten nejpřímočařejší. Zároveň musí tyto scénáře pokrýt pokud možno co největší část aplikace, aby byl výsledek hodnocení relevantní.

U každého scénáře je uveden výchozí bod (stav, ve kterém se uživatel nachází na začátku scénáře), popis a cíl daného scénáře. Výsledkem je pak tabulka, ve které je uvedeno jaký je počet předpokládaných kliknutí myší, psaní na klávesnici a počet těchto akcí dohromady. Případně je také uvedena tabulka, která reprezentuje požadované vstupní parametry a omezení scénáře. Vstupní parametry jsou v hodnotících scénářích pojmenovávány podle písmen abecedy, takže se očekává jeden znak pro zadání názvu parametru.

Scénář č. 1

Vstupním bodem prvního scénáře je moment ihned po otevření aplikace. Tento scénář má za cíl pokrýt prostou kombinaci 2 parametrů a zobrazení výsledku. Nastavení SUT budou ponechána na výchozí hodnotě. Požadované parametry jsou následující:

boolean	boolean
true	true
false	false

Tabulka 5.1: Tabulka vstupních parametrů scénáře č. 1

Scénář č. 2

Vstupním bodem druhého scénáře je moment ihned po otevření aplikace. Tento scénář má za cíl pokrýt prostou kombinaci 3 parametrů a zobrazení výsledku. Nastavení SUT budou ponechána na výchozí hodnotě. Uživatel si před vygenerováním celé toto nastavení vyexportuje. Požadované parametry jsou následující:

boolean	integer	string
true	i > 0 and i < 10	len > 8
false	i < 50	s = "ahoj"

Tabulka 5.2: Tabulka vstupních parametrů scénáře č. 2

Scénář č. 3

Vstupním bodem třetího scénáře je moment, kdy uživatel znovu otevře aplikaci a chtěl by vygenerovat stejnou testovací sadu, kterou nastavil ve druhém scénáři 5.1, kde si nastavení vyexportoval. Ke stávajícím parametrům by chtěl uživatel přidat ještě další parametr datového typu "float". Parametry tedy vypadají následovně:

boolean	integer	string	float
true	i > 0 and i < 10	len > 8	i > 10.5
false	i < 50	s = "ahoj"	i < 20.55

Tabulka 5.3: Tabulka vstupních parametrů scénáře č. 3

Výsledek scénáře č. 1

	aktuální	návrh
myš	7	9
klávesnice	2	2
dohromady	9	11

Tabulka 5.4: Tabulka výsledku scénáře č. 1

U tohoto výsledku lze vidět, že aktuální implementace má nižší počet interakcí, což je však v tomto případě způsobeno tím, že je požadován malý počet parametrů.

Výsledek scénáře č. 2

	aktuální	návrh
myš	14	22
klávesnice	43	22
dohromady	57	44

Tabulka 5.5: Tabulka výsledku scénáře č. 2

Na výsledcích tohoto scénáře lze vidět, že u nastavení s více parametry, které obsahují složitější výrazy již vychází v ohledu interakcí s klávesnicí a myší lépe nově navržené rozhraní. Lehce je navýšeno užívání myši, ale snižuje potřebu používání klávesnice.

Výsledek scénáře č. 3

	aktuální	návrh
myš	24	16
klávesnice	67	10
dohromady	91	26

Tabulka 5.6: Tabulka výsledku scénáře č. 3

Výsledek tohoto scénáře ukazuje, že funkcionalita zabývající se exportováním a importováním testovacího nastavení je přínosná, protože ve staré verzi by uživatel musel celé nastavení vytvářet znovu, kdežto zde lze staré nastavení nahrát a znovu využít s modifikacemi.

5.2 Uživatelské testování

Tato sekce se zaměřuje na vytvoření testovacích úloh. Cílem těchto úloh je prakticky ověřit, zda jsou reální uživatelé schopni orientovat se v nově implementovaném uživatelském rozhraní. Pro testovací účely byli vybrání uživatelé, kteří mají vysoké zkušenosti s používáním počítače, ale s touto aplikací zatím žádné zkušenosti nemají.

Proto byl vytvořen dotazník, ve kterém se nacházejí 4 úlohy. Uživatelé postupně plní jednu úlohu za druhou a zapisují následující informace:

- Jak dlouho pokus trval. Tato informace má za cíl ověřit, zdali se uživatelé postupně učí jak rozhraní používat a jestli má čas klesavou tendenci.
- Ohodnocení od 1 (velmi snadná) do 5 (velmi náročná), indikující, jak moc byla úloha náročná.
- S čím měli největší problém při plnění úlohy. Tato informace ukazuje, kde jsou slabá místa aplikace. Lze tak najít části aplikace, které lze zlepšit.
- Jestli našli při plnění úlohy nějakou chybu. Díky této informaci lze odhalit chyby, které nebyly odhaleny automatickými testy.

5.2.1 Zadání testovacích úloh

V následujících sekcích je vždy uveden popis úlohy, kterou uživatelé dostali. Dále jsou pro každou úlohu uvedeny její výsledky, které vyplnili vybraní uživatelé v dotazníku. Sbírané informace jsou uvedeny výše v sekci 5.2.

Úloha č. 1

Testovací nastavení nechej původní. Vygeneruj testovací sadu, kde budou parametry datových typů integer a string. Parametr typu integer bude mít bloky "větší než 10" a "menší než 20", parametr s datovým typem string by měl mít 3 bloky, které mohou nabývat anglických libovolných slov.

Úloha č. 2

Vygeneruj testovací sadu, kde SUT by se mělo jmenovat "Test", jinak nechej testovací nastavení původní. Budou zde parametry datových typů 2x boolean. Toto nastavení vyexportuj. Proveď znovunačtení stránky a zkus toto vyexportované nastavení znovu importovat. Následně vygeneruj výsledek.

Úloha č. 3

Pojmenujte SUT "Testovací sada". Přidejte parametry datových typů date, time a enum. Don't care hodnoty by měly být randomizovány a vygenerujte tuto sadu pouze s využitím indexů bloků. Sílu T nastav na 3. Datum musí mít bloky "větší než 1.ledna 2020" a "menší než 1.ledna 2024". Čas musí mít bloky "větší než 16:30" a "menší než "17:50". V parametru s datovým typem přidejte 3 bloky s následujícími hodnotami: "pes", "kocka" a "ryba".

Úloha č. 4

Testovací nastavení nechej původní. Přidej 2 parametry typu boolean a vygeneruj výsledek. Následně zkus stáhnout tyto výsledky ve formátu JSON, XML i CSV.

5.2.2 Výsledky zadaných úloh

V této sekci se nacházejí výsledky testovacích úloh, které byly získány z dotazníku. Ke každé sbírané informaci je připravena odrážka, která obsahuje zprůměrovaný výsledek od všech uživatelů. Pod seznamem se získanými informacemi se nachází shrnutí poznatků, které byli danou úlohou získány.

Výsledek úlohy č. 1

- Uživatelům trvalo průměrně 55 sekund, aby tuto úlohu vyřešili.
- Uživatelé průměrně ohodnotili náročnost této úlohy známkou 1,5. To znamená, že úloha byla relativně snadná.
- Uživatelé měli problém s tím, že nelze změnit typ bloku.
- Uživatelé při plnění úlohy nenalezli chybu.

S výsledků této úlohy v dotazníku lze určit, že testujícím uživatelům úloha trvala poměrně dlouho, ale to může být tím, že s touto aplikací neměli dřívější zkušenosti. Nalezeným nedostatkem pak je chybějící editace typu bloku.

Výsledek úlohy č. 2

- Uživatelům trvalo průměrně 40 sekund, aby tuto úlohu vyřešili.
- Uživatelé průměrně ohodnotili náročnost této úlohy známkou 1, což značí, že úloha byla velmi snadná.
- Uživatelé při plnění této úlohy neměli s ničím problém.
- Uživatelé při plnění úlohy nenalezli chybu.

Je zde vidět snížená doba trvání, která je určená i tím, že se uživatelé již naučili základy používání aplikace. Uživatelé žádné problémy při plnění této úlohy nenalezli.

5.2.3 Výsledek úlohy č. 3

- Uživatelům trvalo průměrně 2 minuty, aby tuto úlohu vyřešili.
- Uživatelé průměrně ohodnotili náročnost této úlohy známkou 1,5. To znamená, že úloha byla poměrně snadná
- Uživatelé při plnění této úlohy neměli s ničím problém.
- Uživatelé nalezli chybu, kde při zvolení datového typu enum v parametru a vybrání bloků s typem "Random string value" způsobuje chybu.

Tato úloha uživatelům trvala déle, protože má mnoho obsahu. Chyba, která byla nalezena, byla také řádně opravena.

Výsledek úlohy č. 4

- Uživatelům trvalo průměrně 30 sekund, aby tuto úlohu vyřešili.
- Uživatelé průměrně ohodnotili náročnost této úlohy známkou 1, což značí, že úloha byla velmi snadná.
- Uživatelé při plnění této úlohy neměli s ničím problém.
- Uživatelé při plnění úlohy nenalezli chybu.

Tato úloha byla poměrně snadná, a proto uživatelům ani netrvala dlouho. Žádné problémy v této úloze nebyly nalezeny.

5.3 Testování aplikace

5.3.1 Automatické testy GUI

Tato sekce se věnuje otestování nově implementovaného uživatelského rozhraní. Pro tento účel byly vytvořeny automatické testy grafického uživatelského rozhraní. Testovací sady byly vytvořeny za pomocí knihoven *Jest* a *React Testing Library*.

Testy jsou v případě těchto zvolených knihoven implementovány tak, že lze pro každou komponentu (komponenty jsou popsány v sekci 4.3) vytvořit její vlastní test, který otestuje funkcionalitu této komponenty. Testovací rozsah je 49 testů v 28 testovacích sadách, tedy každá komponenta vytvořená jako část uživatelského rozhraní v aplikaci je otestována.

Test každé komponenty začíná tím, že je vykreslena testovaná komponenta a následně se testují její vlastnosti a funkcionality. Testovací komponentu je třeba také obalit do kontextu, o kterých pojednává sekce 4.2.2, protože data komponent jsou vázána na tyto kontexty. Do kontextů pak lze nahrát výchozí data, která jsou následně propagována do komponent a lze je pak testovat s libovolnými vstupními daty.

Komponenty byly testovány tak, že první bylo vždy otestováno, jestli se dokáže správně vykreslit. Následně většina testovacích sad komponent obsahuje testy, které ověřují, zda správně fungují elementy pro vstupy těchto komponent. Pro parametry a omezení bylo také ověřeno, zda jdou přidat, jestli dostanou správný index a jestli jdou správně smazat. Komponenta jejíž funkcionalita se nepovedla automatickými testy otestovat správně je *FileUpload*. Tato komponenta totiž používá "drag and drop" mechanismus, který není snadné otestovat.

Testy jsou umístěny v adresáři *Tests*. V tomto adresáři je stejná adresářová struktura, jako pro soubory se zdrojovými kódy pro komponenty. Adresářovou strukturu lze vidět zde 4.2.1. Popis textů se nachází vždy v komentáři nad definicí testu. Tyto testy lze považovat za regresní testy, které se využívají pro zachycení potencionálních nově vzniklých chyb při přidávání nových funkcionalit.

5.3.2 Ruční testování

Jedním z nejužitečnějších způsobů testování grafického uživatelského rozhraní je právě ruční testování, protože některé části aplikace téměř nelze otestovat automatickými testy. Proto i v této práci proběhlo ruční testování. Při přidání každé nové funkcionality, byla tato funkcionalita otestována klikáním na jednotlivé elementy a následným vizuálním zkoumáním reakce na tyto akce. Tímto způsobem se povedlo opravit nalezené chyby při vývoji aplikace.

Dalším způsobem ručního testování bylo zadání testovacích úloh pokusným uživatelům. Ti měli do dotazníku u každé zadané úlohy napsat, jestli nenalezli při dané úloze chybu. Tento způsob je popsán v sekci 5.2.

Posledním způsobem ručního testování bylo praktické používání aplikace, kdy bylo vyzkoušeno, zda aplikace opravdu dovede usnadnit tvorbu testovacích dat, které byly následně skutečně využity pro testování jiných aplikací. Tímto způsobem bylo zjištěno, že je opravdu možné tuto aplikaci používat prakticky.

Kapitola 6

Závěr

V této práci bylo vytvořeno webové uživatelské rozhraní pro nástroj Combine implementované v jazyce JavaScript a knihovně React. Toto uživatelské rozhraní umožňuje oproti původnímu prototypu export a následný import testovacích nastavení, zadávání bloků parametrů pomocí kritérií, nebo výběrem hodnot bloků z předdefinované datové sady. Nedostatky původního prototypu byly opraveny díky využití principů použití od Jakoba Nielsena [3].

Možným dalším rozšířením by mohla být například implementovaná možnost přidání datové sady přímo uživatelem, jejíž hodnoty by pak mohl využívat při tvorbě testovací sady. Dále je vhodné přidat možnost přerušení operace generování výsledků. Dalším rozšířením se může nabízet integrace tohoto uživatelského rozhraní s novým vylepšeným nástrojem Combine.

Literatura

[1] BREJCHA, J. Co skrývá uživatelské rozhraní? In: Červenková, A. a Hořava, M., ed. *Uživatelsky přívětivá rozhraní: První sborník o HCI v České republice*. Horava & Associates, 2009, s. 43–52. ISBN 978-80-254-5295-0.

[2] Lei, Y., Kacker, R., Kuhn, D. R., Okun, V. a Lawrence, J. IPOG: A General Strategy for T-Way Software Testing. In: *14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)*. IEEE, Březen 2007, s. 549–556. DOI: 10.1109/ECBS.2007.47. ISBN 0-7695-2772-8.

[3] Nielsen, J. 10 Usability Heuristics for User Interface Design. *Nielsen Norman Group* [online]. 24. dubna 1994. 2024-01-30 [cit. 2024-04-20]. ISSN 1548-5552. Dostupné z: https://www.nngroup.com/articles/ten-usability-heuristics/.

[4] Rodriguez, C., Baez, M., Daniel, F., Casati, F., Trabucco, J. et al. REST APIs: A Large-Scale Analysis of Compliance with Principles and Best Practices. In: Bozzon, A., Cudre Maroux, P. a Pautasso, C., ed. *Web Engineering*. Cham: [b.n.], červen 2016, sv. 9671, s. 21–39. DOI: 10.1007/978-3-319-38791-8_2. ISBN 978-3-319-38790-1.

[5] SAS, M.-U. *MUI* [online]. 2024 [cit. 2024-04-20]. Dostupné z: https://mui.com/.

[6] Týnovský, L. *Specifikace požadavků na software ve formátu případů užití* [online]. Plzeň, 2014. [cit. 2023-12-30]. Bakalářská práce. Západočeská univerzita v Plzni, Fakulta aplikovaných věd. Dostupné z: https://otik.zcu.cz/bitstream/11025/13544/1/A10B0357Pbp.pdf.

[7] Yu, L., Lei, y., Nourozborazjany, M., Kacker, R. a Kuhn, D. An Efficient Algorithm for Constraint Handling in Combinatorial Test Generation. In: *2013 IEEE Sixth International Conference on Software Testing, Verification and Validation*. IEEE, Březen 2013, s. 242–251. DOI: 10.1109/ICST.2013.35. ISBN 978-1-4673-5961-0.

[8] Červinka, R. *Combine's Guide* [online]. Brno: Fakulta informačních technologií VUT v Brně [cit. 2023-12-30]. Dostupné z: https://combine.testos.org/help.

[9] Červinka, R. *Asistent pro generování testovacích scénářů* [online]. Brno, 2018. [cit. 2023-12-30]. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Dostupné z: https://www.vut.cz/studenti/zav-prace/detail/114616.

Příloha A

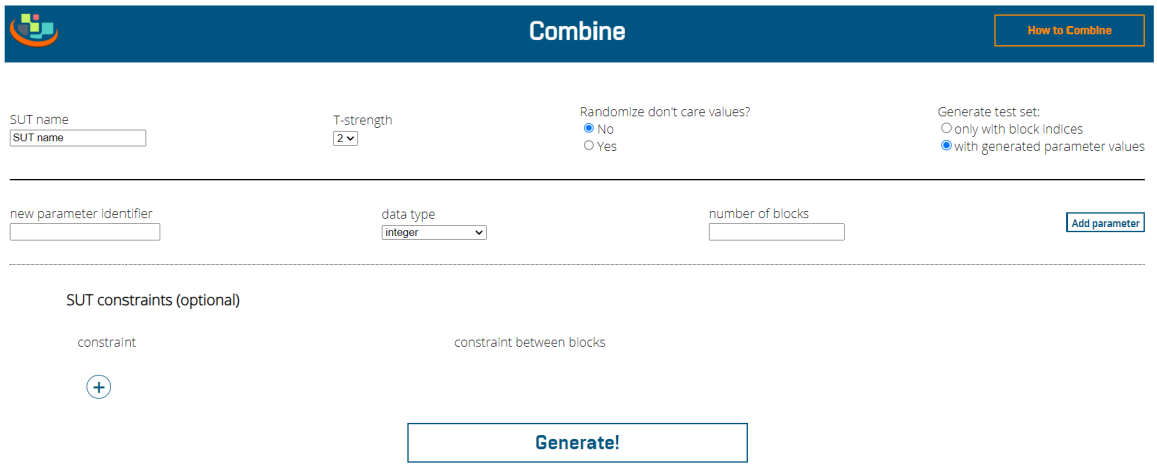
Obsah přiloženého média

Přiložené médium obsahuje následující soubory a složky:

- složku **backend**, která obsahuje soubory se zdrojovým kódem pro serverovou část aplikace:
  - soubor **api.py**, ve kterém se nachází serverová část aplikace
  - složku **DataSets**, ve které jsou testové soubory s jednotlivými datovými sadami
- složku **combine_ui**, která obsahuje soubory se zdrojovými kódy pro klientskou část aplikace:
  - složku **node_modules**, která obsahuje závislosti aplikace v *Node.js*
  - složku **public**, která obsahuje soubory pro statické hodnoty
  - složku **src**, která obsahuje složky a soubory se zdrojovými kódy pro testy a jednotlivé komponenty, ze kterých se skládá webové uživatelské rozhraní
  - soubor **.gitignore**, který definuje soubory, které by neměly být stopovány nástrojem GIT
  - soubor **.prettierrc.json**, který určuje formátování textu se zdrojovými kódy
  - soubor **package.json**, který obsahuje metadata o projektu a obsahuje závislosti projektu
  - soubor **package-lock.json**, který určuje verze závislostí tak, aby byly konzistentní pro každého vývojáře
- soubor **README.md**, který obsahuje popis projektu, návod na spuštění a závislosti
- soubor **BP_xnenut01.pdf**, který obsahuje písemnou zprávu
- ZIP složku **BP_xnenut01.zip**, která obsahuje zdrojový tvar písemné zprávy

Příloha B

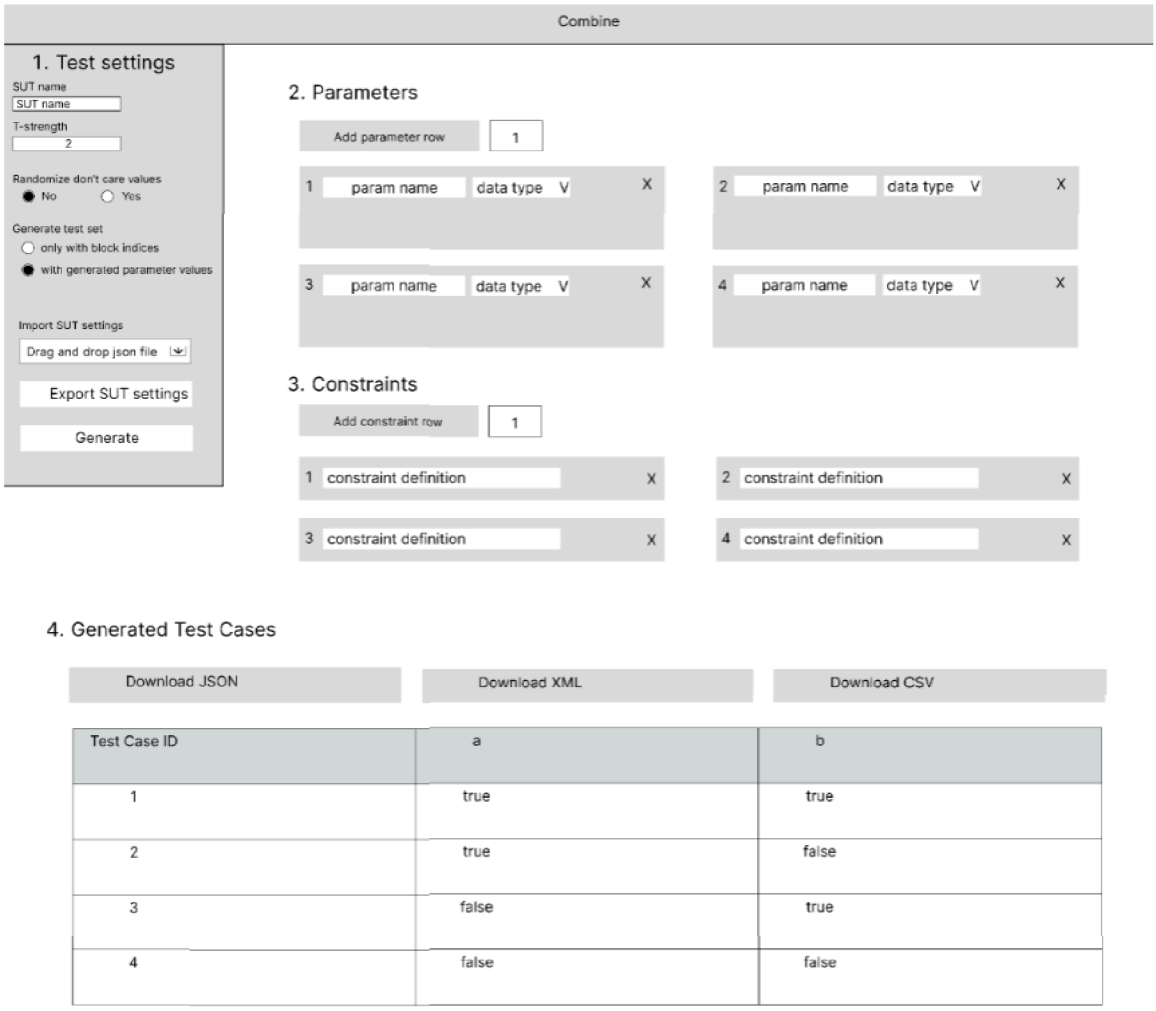
Aktuální uživatelské rozhraní



Obrázek B.1: Aktuální podoba grafického uživatelského rozhraní nástroje Combine

Příloha C

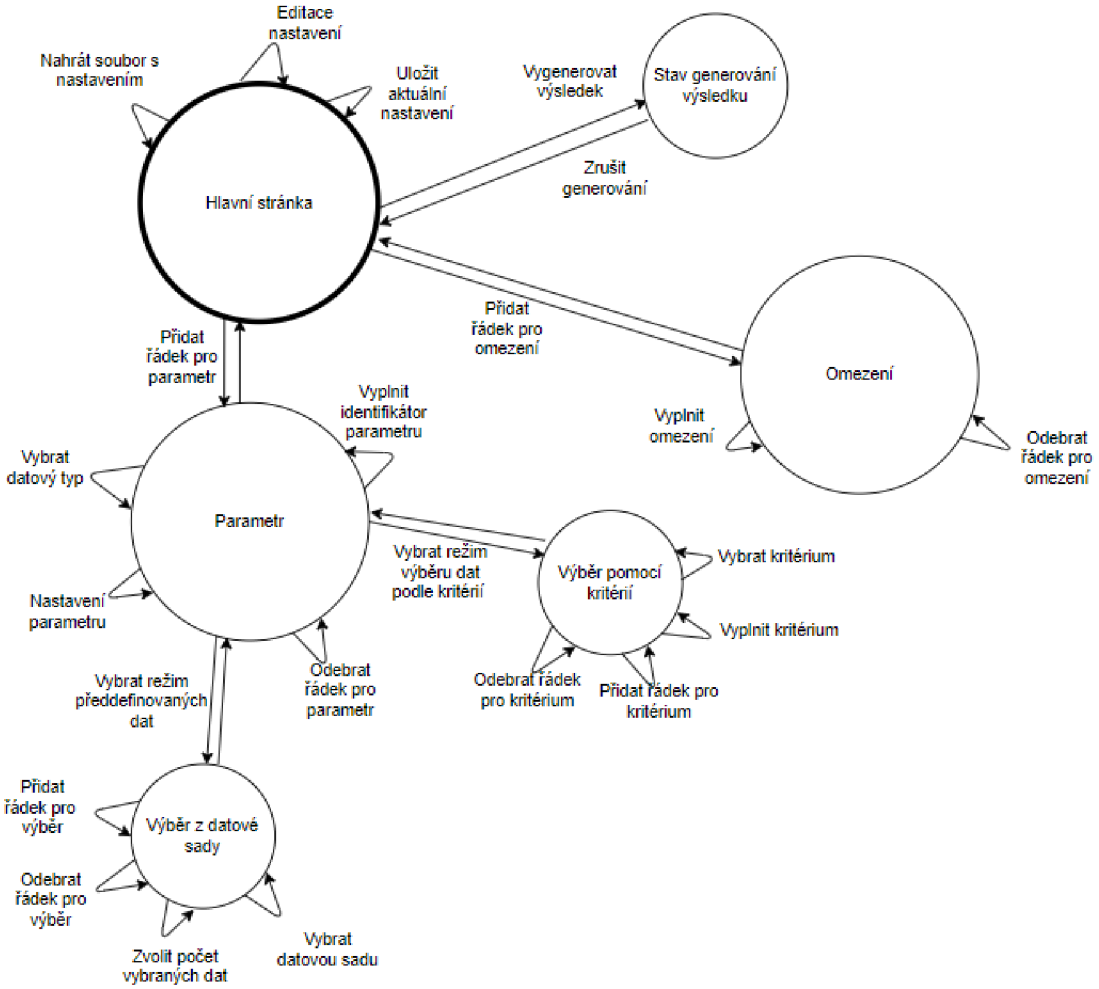
Návrh nového uživatelského rozhraní



Obrázek C.1: Wireframe zobrazující rozložení prvků

Příloha D

Diagram zaměření pozornosti



Obrázek D.1: Diagram zaměření pozornosti uživatele na určitou oblast uživatelského rozhraní

Příloha E

Nové uživatelské rozhraní

Obrázek E.1: Nově vytvořené uživatelské rozhraní

Příloha F

Uživatelská příručka

F.1 Závislosti

F.1.1 Klientská část

- node.js: 21.6.2
- npm: 10.2.4
- emotion/react: 11.11.4
- emotion/styled: 11.11.0
- fontsource/roboto: 5.0.12
- mui/icons-material: 5.15.14
- mui/material: 5.15.14
- testing-library/jest-dom: 5.17.0
- testing-library/react: 13.4.0
- testing-library/user-event: 13.5.0
- react: 18.2.0
- react-dom: 18.2.0
- react-dropzone: 14.2.3
- react-scripts: 5.0.1
- styled-components: 6.1.8
- web-vitals: 2.1.4

F.1.2 Serverová část

- Flask 3.0.0
- Flask-Cors 4.0.0

F.2 Jak spustit aplikaci?

1. Naklonujte nebo stáhněte projekt z repozitáře [1].
2. Je třeba nainstalovat požadovanou verzi *npm*, která je uvedena zde F.1.1.
3. Spusťte příkaz "npm install"ve složce *combine_ui* pro stažení všech potřebných závislostí.
4. Nainstalujte *Python 3*
5. Nainstalujte *pip*
6. Nainstalujte *Flask* a *Flask-Cors* s verzemi uvedenými zde F.1.2.
7. Spusťte program *api.py* v složce *backend* příkazem "python api.py".
8. Spusťte příkaz "npm start" ve složce *combine_ui* pro spuštění aplikace.
9. Pokud chcete využít samotného nástroje Combine, musíte si stáhnout nebo naklonovat projekt z repozitáře [2]. Pro instalaci nástroje použijte přiložené *README.md*, které je součástí projektu. Je možné, že bude ještě třeba nastavit CORS v nástroji Combine. *N*yní lze plnohodnotně používat nástroj Combine s novým uživatelským rozhraním.

F.3 Jak přidat datovou sadu?

1. Nachystejte si textový soubor s datovou sadou, kde každý prvek datové sady bude na novém řádku.
2. V souboru *src/Data/parameter.json* si vyberte, ke kterému datovému typu chcete datovou sadu přidat.
3. Do pole vybraného datového typu přidejte stejný název, který má váš nachystaný textový soubor s datovou sadou.
4. Nachystaný textový soubor s datovou sadou vložte do složky *backend/DataSets*.
5. Nyní je možné využívat vámi přidanou datovou sadu u vybraného datového typu.

F.4 Jak spustit testy?

Pro spuštění testů je třeba napsat příkaz "npm test" ve složce *combine_ui*.

[1] `https://pajda.fit.vutbr.cz/testos/neocombine`
[2] `https://pajda.fit.vutbr.cz/testos/combine`