Univerzita Pardubice
Fakulta elektrotechniky a informatiky

Návrh a realizace nástroje pro testování REST API

Michal Motyčka

Bakalářská práce
2022

Univerzita Pardubice
Fakulta elektrotechniky a informatiky
Akademický rok: 2021/2022

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Michal Motyčka**
Osobní číslo:	**I19124**
Studijní program:	**B2646 Informační technologie**
Studijní obor:	**Informační technologie**
Téma práce:	**Nástroj pro testování REST API**
Zadávající katedra:	**Katedra informačních technologií**

Zásady pro vypracování

Cílem práce bude vytvořit aplikaci pro automatické testování HTTP API. Aplikace umožní testování pomocí ručně zadaných dotazů. Dále bude podporovat automatické generování testovacích dotazů podle zadaných preferencí. V teoretické části práce student popíše princip testování API. Text práce bude kromě samotné problematiky obsahovat i přehled použitých technologií, rešerši o existujících alternativách a analýzu s realizací aplikace.

Rozsah pracovní zprávy: **min. 30 stran**
Rozsah grafických prací:
Forma zpracování bakalářské práce: **tištěná/elektronická**

Seznam doporučené literatury:

RILEY, Tim a Adam GOUCHER. Beautiful testing. Sebastopol, Calif.: O'Reilly, 2010. Theory in practice (Sebastopol, Calif.). ISBN 978-0596159818.
BUREŠ, Miroslav, Miroslav RENDA, Michal DOLEŽEL, Peter SVOBODA, Zdeněk GRÖSSL, Martin KOMÁREK, Ondřej MACEK a Radoslav MLYNÁŘ. Efektivní testování softwaru: klíčové otázky pro efektivitu testovacího procesu. Praha: Grada, 2016. Profesionál. ISBN 978-80-247-5594-6.
ROUDENSKÝ, Petr a Anna HAVLÍČKOVÁ. Řízení kvality softwaru: průvodce testováním. Brno: Computer Press, 2013. ISBN 978-80-251-3816-8.

Vedoucí bakalářské práce: **Ing. Jan Merta**
Katedra softwarových technologií

Datum zadání bakalářské práce: **17. prosince 2021**
Termín odevzdání bakalářské práce: **13. května 2022**

L.S.

Ing. Zdeněk Němec, Ph.D. v.r.
děkan

Ing. Jan Panuš, Ph.D. v.r.
vedoucí katedry

V Pardubicích dne 28. února 2022

Prohlašuji:

Tuto práci jsem vypracoval samostatně. Veškeré literární prameny a informace, které jsem v práci využil, jsou uvedeny v seznamu použité literatury.

Byl jsem seznámen s tím, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorský zákon, zejména se skutečností, že Univerzita Pardubice má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona, a s tím, že pokud dojde k užití této práce mnou nebo bude poskytnuta licence o užití jinému subjektu, je Univerzita Pardubice oprávněna ode mne požadovat přiměřený příspěvek na úhradu nákladů, které na vytvoření díla vynaložila, a to podle okolností až do jejich skutečné výše.

Beru na vědomí, že v souladu s § 47b zákona č. 111/1998 Sb., o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších předpisů, a směrnicí Univerzity Pardubice č. 9/2012, bude práce zveřejněna v Univerzitní knihovně a prostřednictvím Digitální knihovny Univerzity Pardubice.

V Pardubicích dne 12. 5. 2022

Michal Motyčka

Poděkování

Děkuji vedoucímu mé bakalářské práce, panu Ing. Janu Mertovi za odborné vedení, ochotu a cenné rady, které umožnily kvalitnější zpracování bakalářské práce. Dále bych chtěl poděkovat své rodině za morální podporu v průběhu mého studia.

Anotace

Cílem bakalářské práce je vytvořit aplikaci pro testování RESTful API pomocí HTTP požadavků. V teoretické části je věnovaná pozornost technologiím, které daná aplikace využívá a základním pojmům. V praktické části je popsán vývoj samotného testovacího softwaru. Aplikace umožní manuální zadaní testovacích dotazů, a také automatické generování testovacích dotazů dle zadaných preferencí.

Klíčová slova

Webová aplikace, HTTP, RESTful API, testování, JSON, URL, generování dotazů, testovací data

Title

Design and implementation of tool for testing REST API

Annotation

The goal of the bachelor thesis is to create an application for testing RESTful API using HTTP requests. The theoretical part pays attention to the technologies that the application uses and its basic concepts. Practical part is description on how the testing software was developed. The application allows you to enter test requests manually, as well as automatically generate test requests according to the entered preferences.

Keywords

Web application, HTTP, RESTful API, testing, JSON, URL, query generation, test data

OBSAH

SEZNAM OBRÁZKŮ

SEZNAM ZKRATEK

AJAX	Asynchronous JavaScript and XML
AOT	Ahead-of-time
API	Application Programming Interface
BE	Back End
CI/CD	continuous integration / continuous delivery
CIL	Common Intermediate Language
CLR	Common Language Runtime
CRUD	Create, Read, Update, Delete
EUPL	European Union Public Licence
FE	Front End
GC	Garbage collector
HATEOAS	Hypermedia as the engine of application state
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
IoC	Inversion of control
IOT	Internet of things
JIT	Just-in-time
JSON	JavaScript Object Notation
MIME	Multipurpose Internet Mail Extensions
MVVM	Model-View-ViewModel
REST	Representational state transfer
SOA	Service Oriented Architecture
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
URI	Uniform Resource Identifier
WPF	Windows Presentation Foundation
WSDL	Web Services Description Language
XAML	Extensible Application Markup Language

ÚVOD

Webové technologie jsou momentálně velmi populární. Z této popularity vzniklo velké množství konceptů zobrazování dat uživatelům. Nejedná se jen o webové stránky s uživatelským rozhraním, ale i koncové body, které přijímají nebo poskytují pouze čistá data bez ovládacích a grafických prvků. Tato data jsou zpřístupňována pomocí přístupových bodů a slouží aplikacím, které data využívají pro své účely a přístup přes grafické rozhraní na rozdíl od uživatelů nepotřebují. Bývá zvykem, že těchto přístupových bodů je více, každý má odlišné vlastnosti a sdružení těchto bodů vzniká rozhraní, které nazýváme API.

Při vývoji API je nutné průběžné testování jeho funkčnosti. Testování aplikace je neodmyslitelnou součástí jejího vývoje, proto se ve společnostech objevila dedikovaná pozice právě pro tuto činnost. Je-li aplikováno důsledné testování je zaručena vyšší kvalita výsledného produktu. Každé odvětví se vyvíjí nebo zvyšuje svoji efektivitu a u vývoje softwaru tomu není jinak. Důraz je kladen na zautomatizování čím dál většího množství úkonů, a proto se můžeme setkat se s automatizovanými testy. Cílem této práce je vytvořit aplikaci, která má ulehčit práci nejen testerům, ale i vývojářům. Ulehčení práce bude pomocí testovací aplikace, kde si scénáře uživatel ručně vytvoří nebo jsou vygenerovány na základě zadané definice API.

První kapitola se zabývá definicí testování, principem testovaní API, analýzou stávajících řešení a využití technologií při realizaci výsledné aplikace. První podkapitola je zasvěcena definici samotného testování. V druhé podkapitole je vysvětlen princip testování API. Následuje část, která analyzuje již existující nástroje pro testování, které jsou hojně produkčně využívané. Poslední podkapitola představuje definici pojmů a využité technologie při tvorbě aplikace v praktické části.

Druhá kapitola je věnována návrhu a realizaci nástroje pro testování. V první části druhé kapitoly jsou popsány všechny funkční a nefunkční požadavky. Následující podkapitola je zaměřena na popis využitých datových struktur. Poslední podkapitola je věnována popisu uživatelského rozhraní.

1 TESTOVÁNÍ API

V této kapitole se čtenář dozví definice testování a jejich principy použití. Dále je zde zpracovaná analýza již existujících specializovaných nástrojů, které jsou pro testování používány. Poslední podkapitola je věnována technologiím, které byly využity při tvorbě testovacího nástroje.

1.1 Definice testování

S.M.K Quadri a Sheikh Umar Faooq ve svém článku [10] definují testování: *„Softwarové testování je aktivita, která je zaměřena na vyhodnocování kvality programu pro nalezení defektů a problémů.“*

Podle autorů knihy SWEBOK [11] je softwarové testování složeno z konečné množiny testovacích scénářů, které jsou využity pro dynamické ověření chování softwaru. Díky tomuto ověřování má dojít k nalezení chyb a problémů, které by mohly při běhu aplikace nastat.

1.1.1 Druhy testování

Testování lze rozdělit do dvou základních kategorií podle způsobu realizace na manuální a zautomatizované. Manuální testování je prováděno zpravidla testerem, který ručně provádí testy a ověřuje, zda se program chová tak, jak je předpokládáno. Oproti tomu zautomatizované testování je prováděno specializovaným softwarem, který z předem vytvořených scénářů porovnává reálné výsledky s očekávanými. Pokud jsou automatické testy vytvořeny správně, zpravidla bývají přesnější, jelikož při opakování se daný proces chová identicky. Oproti tomu ruční testování může být odlišné důsledkem lidské chyby. Další výhodou automatického testování je, že šetří lidské zdroje, jelikož vytvořený test stačí pouze spustit a sám se vyhodnotí. Zároveň bývá často rychlejší nebo minimálně stejně rychlý. [4, 8]

Další rozdělení testování může být dle znalosti vnitřní funkcionality daného systému, přesněji přístupu ke zdrojovým kódům daného systému. Jedná se o rozdělení takzvaných white box a black box. Bílá skříňka je, když je přístupný zdrojový kód a tester je schopen přečíst vnitřní funkcionalitu jednotlivých komponent. Zatímco u černých skříněk neznáme vnitřní strukturu a tester je vázaný jen na vstupní a výstupní data z daného systému. [30]

Další pohled na rozdělení testování je dle modelu kvality FURPS, který byl vyvinut společností Hewlett-Packard. Tento název modelu je složen z počátečních písmen jednotlivých kritérií functionality, usability, reliability, performance a supportability. Funkčnost popisuje, zda vyvinutý systém odpovídá všem funkčním požadavkům. Následuje použitelnost, která specifikuje uživatelskou přívětivost k danému systému z pohledu uživatelského rozhraní a dokumentace. Další kritérium je věnováno spolehlivosti daného systému, kde jsou pozorovány parametry systému

a porovnávají funkčnost, zda daný systém pracuje stejně ve všech případech, počet chyb nebo časové období mezi jednotlivými chybami. Výkonnost je zaměřena na odezvu systému za určitých okolností. Příkladem těchto okolností může být zahlcení systému, nebo jeho vyprázdnění, kdy daný systém nic nezpracovává. Posledním kritérium se zabývá škálovatelností výsledného systému a samotnou udržitelností včetně jeho testovatelnosti. [30]

Posledním zmíněným rozdělením testování je dle jednotlivých fází vývoje systému. Mezi tyto fáze patří následující druhy testů. Jednotkové testy jsou zaměřeny na otestování samotné implementace jednotlivých funkčností. Následují modulové testy, které ověřují funkčnost navržených modulů. Zpravidla se liší v rozsahu od jednotkových testů, kdy modulové testy testují provázanost jednotlivých funkčních bloků například knihoven. Jednotkové a modulové testy jsou zautomatizované a jsou realizovány zpravidla programátory. Dále jsou vytvářeny testy integrační, které ověřují provázanost modulů v daném subsystému. Tyto integrační testy již zpravidla přebírají samotní testeři. Následuje funkční testování, kde se ověřuje, zda daný systém/podsystém funguje podle zadaných specifikací uvedených v požadavcích. Další fází je systémové testování, které je realizováno na celém systému. V poslední řadě jsou akceptační testy, které jsou realizovány zákazníkem. Pomocí nich si zákazník ověří funkcionalitu daného systému a zjistí, zda systém splňuje jeho očekávání. [4]

1.2 Princip testování API

V první řadě je potřeba zmínit, že testování webového API je založeno na klient-serverové komunikaci, kde klient odesílá HTTP požadavky na webový server, který je zpracuje a vrací data. K testování webového API je zapotřebí software, který dokáže odesílat jednotlivé HTTP požadavky na server. Samotný webový prohlížeč může být základní nástroj pro manuální testování, ale jsou zde velké limitace v jeho použití. Pro odstranění těchto limitací je vhodné využít specializované nástroje, které často i podporují automatické testování. Podoba těchto nástrojů není nijak definována, může se jednat o rozšíření do prohlížeče, webovou stránku, nebo těžkého klienta nainstalovaného v operačním systému.

Dle článku Automated Specification-Based Testing of REST APIs [6] lze testování API rozdělit do šesti fází. Jedná se o validaci WSDL případně jiného druhu specifikace, jednotkové testování, funkční testování, regresivní testování, zátěžové testování a ověření dodržení standardů. Dále je zde zmíněno, že lze testovat i na tzv. černé skříncе, to je případ, kdy není zapotřebí znát vnitřní architekturu systému, ale předmětem ověřování funkčnosti jsou vstupy a výstupy z daného systému.

Tato práce se zabývá vytvořením nástroje pro výše zmíněné funkční testování. V praxi se jedná o vytvoření scénářů, které mohou obsahovat jednotlivé požadavky nebo i skupinu utvořenou z více požadavků, u kterých ověřujeme odpovědi z API. Z odpovědí jsou důležité zejména stavové kódy nebo data, které API zasílá zpět.

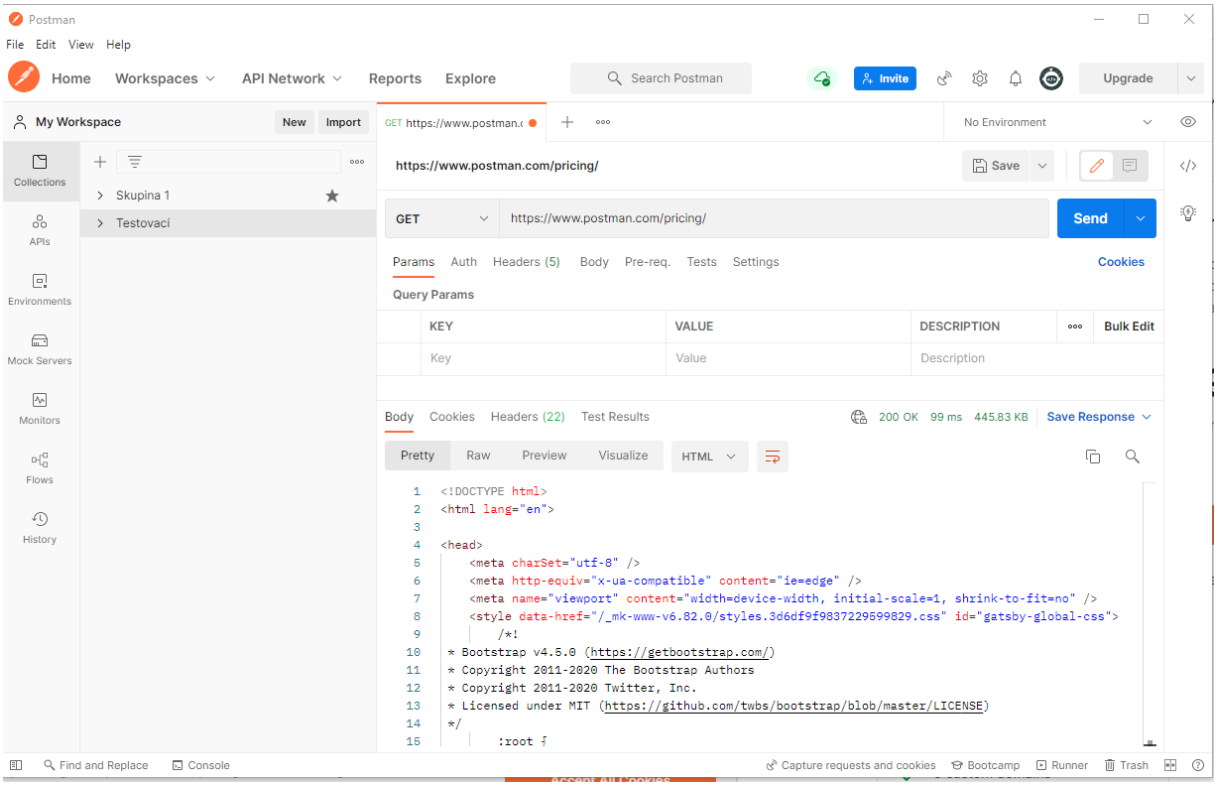
Příkladem může být požadavek, který dle zaslaných parametrů vytvoří a vloží objekt do systému. Pomocí odpovědi z webového server nebo dalším zaslaným požadavkem, který vrátí daný objekt, lze ověřit, že se daný nový objekt skutečně a validně vytvořil.

1.3 Analýza stávajících řešení

Analýza obsahuje rešerši populárních nástrojů pro testování webových API.

1.3.1 Postman

Jedná se o proprietární nástroj pro vývoj a testování webových API. Pro individuální použití nebo pro malé týmy lze použít omezenou licenci, která je zcela zdarma. Pro větší týmy se jedná již o placenou službu, která rozšiřuje funkčnosti daného nástroje dle předplaceného plánu. [2]



Obrázek 1 – Aplikace Postman

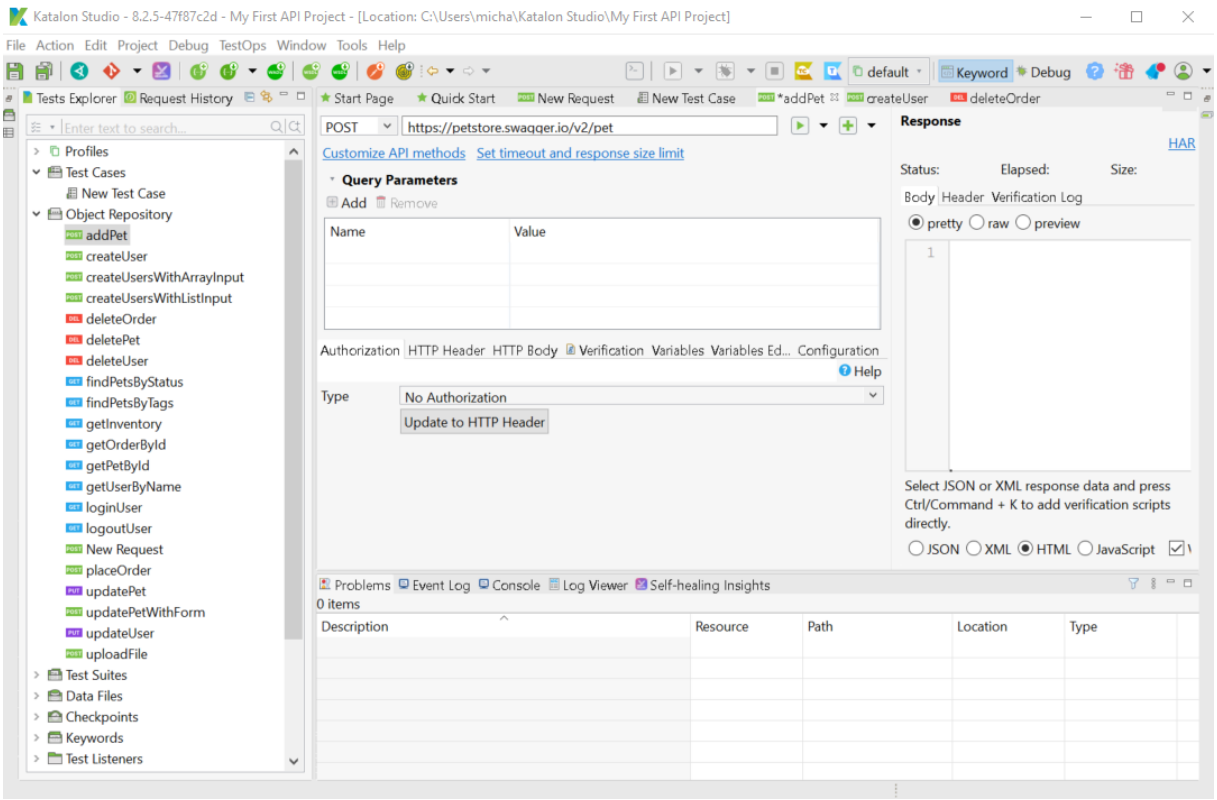
Aplikace dovoluje vytvářet HTTP, WebSocket a gRPC požadavky. U tvorby HTTP požadavku si uživatel vybere typ metody, nadefinuje hlavičku, vyplní URL adresu a případně přidá do těla požadavku data. Následně je schopen daný požadavek odeslat na webový server a otestovat jeho

funkčnost. Jednotlivé dotazy je možné seskupovat do hierarchických skupin, které zlepšují přehlednost uživatelům softwaru. Postman podporuje i importování požadavků přímo z prohlížeče za pomocí schránky pro kopírování ve formě cURL příkazu. Aplikace podporuje i vytváření kódu za pomocí JavaScriptu, který slouží pro automatické testování dotazů. Další výhodou je možnost importovaní API specifikace dle OpenAPI. Aplikace Postman je zobrazena na obrázku 1.

1.3.2 Katalon Studio

Katalon Studio je specializovaný nástroj pro testování webových služeb. Studio je pro individuální použití zcela zdarma, ale pro větší týmovou práci se jedná již o placenou službu. Nevýhodou této služby je znatelně vyšší cena licence oproti výše zmíněnému Postmanu. [3]

Oproti Postmanu se zde nachází širší podpora importování specifikací API. Můžeme zde importovat OpenAPI 3.0, OpenAPI 2.0 dříve Swagger specifikace, WADL, WSDL a projekt z aplikace SoapUI. Hlavní výhodou tohoto nástroje je velká podpora testování. Nástroj podporuje automatické testovaní API typu REST a SOAP, dále také nabízí UI testování. Automatické testy lze naprogramovat ve skriptovacím jazyku Groovy, který je velmi podobný Javě. Součástí nástroje je také debugger, který znatelně zjednodušuje práci při vytváření nových testů. Katalon Studio je zachyceno na obrázku 2.

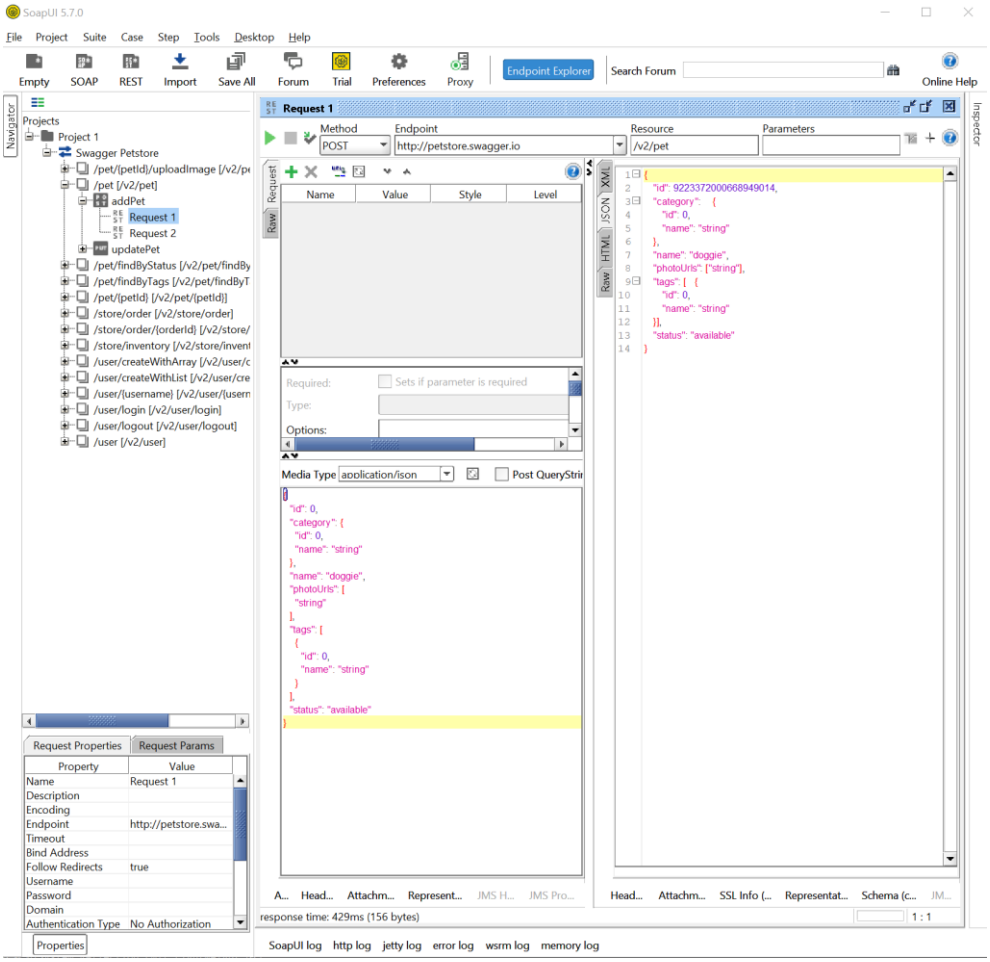


Obrázek 2 – Katalon Studio

1.3.3 SoapUI

Další nástroj pro testování je otevřený software SoapUI, který je šířen pod open source licencí EUPL a zaštiťuje jej společnost SmartBear. Jedná se o nástroj, který je vhodný pro testovaní SOAP a REST API. [12] Nástroj podporuje importování kolekcí z Postmana, dále je zde také možné přidat či uložit specifikaci OpenAPI. Samotná aplikace je zachycena na obrázku 3.

SoapUI sice podporuje automatické testovaní pomocí scriptů, nicméně SmartBear vyvíjí i placenou nástavbu toho produktu ReadyAPI. ReadyAPI je rozdělen do 3 placených modulů, které začínají v době psaní této práce na 755 EUR za jeden rok. Tyto moduly poskytují vyšší flexibilitu testerům. Jedná se například o generátory testovacích dat, CI/CD automatizaci nebo dokonce i vytváření automatických testů umělou inteligencí. [12]



Obrázek 3 – SoapUI

1.4 Technologie pro tvorbu aplikace

Tato podkapitola je věnována technologiím, které souvisejí s realizací testovacího nástroje níže. Čtenář se zde dozví informace o HTTP protokolu, RESTful API, .NET, WPF a dalších použitých technologiích.

1.4.1 HTTP

Hypertext Transfer Protocol je internetový protokol sloužící pro přístup k webovým technologiím. Samotný protokol není zabezpečen. Proto vzniklo rozšíření HTTPS, které využívá SSL nebo TLS. HTTP a HTTPS zpravidla využívají standardizované TCP porty 80 pro nezabezpečenou variantu a 443 pro zabezpečenou. Zároveň se jedná o bezestavový protokol, pro vynucení zapamatování stavů je nutné využít externí řešení, kterými jsou například cookies.

Počátky toho protokolu lze nalézt již na počátku devadesátých let minulého století, kdy první verze 0.9 obsahovala jen metodu GET, která vrátila hledaná data. Druhá verze tj. 1.0 byla vydána záhy, která tento standard rozšířila o nové metody POST a HEAD, podporu MIME, návratové kódy a hlavičky. [13] Pro tuto práci je důležitá až verze 1.1 vydaná 1997, kde Roy Fielding jakožto spoluautor HTTP standardu paralelně při vývoji této verze definoval architekturu REST API ve své disertační práci. [9, 13]. Tato verze přidala mnoho vylepšení, jako například podporu virtuálních hostů, kteří dovolovali hostovat více webových serverů pod jednou IP adresou. Dále zde byly přidány nové metody DELETE, PUT, PATCH, TRACE, CONNECT a OPTIONS, které jsou hojně využívány ve výše zmíněném REST API. Také zde byla přidaná funkcionalita, která udržuje spojení mezi serverem a klientem pro rychlejší komunikaci a stahování dat. Poslední revize této verze byla vydaná v červnu roku 2014 pod RFC 7230. [13]

HTTP/2 standard byl publikován v květnu 2015 pod RFC 7540. [14] Hlavní myšlenkou této nové verze bylo zrychlit webový protokol a minimalizovat zátěž přenosu dat. Proto byl zaveden přenos dat binárním způsobem a kompresí hlaviček pomocí jednoho proudového kanálu. Zároveň zde byly přidány ochrany proti zahlcení webového serveru. [14, 15]. Momentálně je ve vývoji nový standard HTTP/3, který by měl přinést komunikaci přes UDP za pomocí technologie QUIC vyvinutý společností Google. [16] Dle dat W^{3}Techs [17] při psaní té práce podporuje HTTP/2 46,5 % a HTTP/3 24.6 % všech webů a to i přes fakt, že HTTP/3 standard nebyl ještě dokončen.

1.4.2 RESTful API

V roce 2000 Roy Fielding ve své disertační práci [9] definoval REST, tímto činem změnil pohled na design architektur webových technologií. Daná architektura musí splňovat následné podmínky [5, 9]:

1. Klient-serverová architektura
    - Architektura systému musí být navržená jako server-klientově orientovaný systém. Tato orientace zaručuje oddělení funkcionality klienta a serveru, kteří pracují zcela odlišně, ale komunikují spolu přes jednoznačně definované webové rozhraní. [5, 9]
2. Jednotné rozhraní
    - Jedná se o sadu čtyř pravidel, které jsou využity při návrhu RESTful API. První pravidlo je jednoznačná identifikace prostředku. Jedná se o to, že každý prostředek musí mít unikátní identifikátor tzv. URI. Další pravidlo se nazývá manipulace prostředků pomocí reprezentace. Klient nepracuje s prostředkem, ale s její reprezentací a praktikuje na ní CRUD. Reprezentace může být v podobě JSON, XML, HTML a další. Předposlední pravidlo jsou sebe popisující zprávy. Každá zpráva s sebou nese dostatečné množství informací, které popisují možnosti zpracování dané zprávy. Posledním pravidlem je HATEOAS a určuje stav aplikace. Další odkazy na stavy lze nalézt v odpovědi ze serveru. [5, 7, 9]
3. Bezstavovost
    - Každý požadavek musí obsahovat veškeré informace k jejímu zpracování, tzn. v URL adrese, hlavičkách nebo odesílaných datech. Webový server si nepamatuje stavy přechozích požadavků, proto se tento princip nazývá bezstavovost (anglicky stateless). [5, 9]
4. Vrstevnatelný systém
    - Systém musí podporovat vkládání dalších vrstev jako jsou podpora proxy serverů, šifrování, cache a další. Tyto vrstvy zpravidla zajišťují snižovaní odezvy, zvyšování zabezpečení nebo rozložení zátěže. [5, 9]

5. Podpora Cache

    - Systém musí podporovat využití mezipaměti – cache, tato funkcionalita dokáže snížit režii webového serveru, jelikož na některé požadavky lze využít mezipaměť, jedná se primárně o data, které jejichž doba validity bývá vysoká. Jako příklad dat s vysokou dobou validity lze uvést například obrázky, které nebývají měněny často. Daná cache může být kdekoliv na cestě síťovým prostředím mezi serverem a cílovým klientem. Dále je nutné každý požadavek klienta označit, zda si přejeme cache využít či nikoliv. [5, 7, 9]

6. Code-On-Demand

    - Webový server může poskytnout spustitelný kód, který rozšíří nebo pozmění funkcionalitu na straně klienta. Může se jednat například o kód napsaný v Javascriptu. Jedná se o jedinou volitelnou podmínku. [5, 9]

1.4.3 OpenAPI

Jedná se o otevřenou specifikaci, která slouží pro popis RESTful API. Hlavním cílem specifikace je porozumění API bez přístupu do zdrojových kódu.

Historie OpenAPI sahá do roku 2011, kdy byla poprvé publikována verze 1.0 pod názvem Swagger Specification společností SmartBear. Koncem roku 2015 byl projekt darován sdružení vývojářů OpenAPI Initiative. Toto darování proběhlo již na vyvinuté verzi 2.0 a tato specifikace byla přejmenována na OpenAPI. Sice byl projekt oficiálně přejmenován, ale dodnes se hojně využívá starý název Swagger, především tedy mezi vývojáři, pokud se mluví o verzi 2.0. V druhé polovině roku 2017 byla vydaná verze 3.0. Momentálně nejnovější verze je 3.1, která byla vydaná v únoru 2021. [18]

SmartBear sice daroval Swagger specifikaci, ale stále vyvíjí otevřené nástroje, které jsou rozšířeny o podporu nových verzí OpenAPI specifikace. Jedná se o Swagger Editor, který slouží k návrhu, popisu a dokumentaci API. Dále nástroj Swagger Codegen, který se používá ke generování klientských a serverových zdrojových kódu z definovaného API pro různé programovací jazyky. Nebo Swagger UI, který je určen pro vizualizaci definice API v přehledné formě. [20]

1.4.4 .NET

Platforma .NET je otevřenou platformou pro vývoj aplikací. Spadá pod licence MIT a Apache 2. Zároveň je vyvíjená společností .NET Foundation, která byla založena společností Microsoft. Tato platforma je podporovaná na systémech Windows, Linux, Android macOS, iOS a dalších. Dále podporuje nejrozšířenější procesorové architektury, kterými jsou x86, x64, ARM32 a ARM64. Platformu .NET lze využít na velké spektrum typů aplikací. Může se jednat o webové služby, mobilní a desktopové aplikace, IOT, hry, strojové učení a mnoho dalších typů softwaru. [1]

Počátky této platformy lze nalézt na začátku druhého tisíciletí, kdy vyšla první verze. Z počátku se jednalo jen o technologii určenou pro Windows, která je známa pod označením .NET Framework. V průběhu času se daná platforma vyvíjela a přidávala nové moderní technologie, jako jsou například metodiky pro asynchronní programování, podpora paralelizace a mnoho dalších. [1, 21]

K první multiplatformní implementaci paradoxně došlo z iniciativy komunity, která vytvořila vlastní otevřenou implementaci CIL, který byl vydán jakožto otevřený standard. Tato implementace byla pojmenovaná Mono. Vývoj zaštiťoval Ximian, později Novell a momentálně Xamarin, který je v tuto dobu dceřinou společností Microsoft. Mono přineslo s vlastními knihovnami podporu na operačních systémech typu UNIX, Linux, ale i Windows. Roku 2016 byl vydán .NET Core, který byl vytvořen od nuly s cílem vytvořit platformu, která podporuje mnoho operačních systémů a přepíše aktuální .NET Framework. [1, 21] Toto přepsání se úspěšně podařilo s příchodem .NET 5, která vyšla koncem roku 2020. Momentálně vydaná verze je .NET 6 a verze 7, která je momentálně v předběžném přístupu by měla být vydána v třetím kvartálu roku 2022. [22]

K dnešnímu dni platforma podporuje tři programovací jazyky. Jsou to jazyky C#, funkcionální F# a Visual Basic. Zdrojové kódy jakéhokoliv výše zmíněného programovacího jazyka jsou zkompilovány do mezijazyka CIL, o kterém můžeme říct, že se jedná o jazyk principově podobnému bytecodu v Javě. Při spuštění aplikace je spuštěn CLR, který za pomocí JIT nebo AOT kompilátorů překládá CIL na strojový kód, který je následně spuštěn na cílovém počítači. [1]

1.4.5 C#

C# je vysokoúrovňový objektově orientovaný programovací jazyk vycházející z C++ a Javy. Právě proto je syntaxe velmi podobná těmito dvěma jazykům. Jedná se o nejpoužívanější programovací jazyk na platformě .NET. Byl vydán s první verzí této platformy. C# je v současné době jedním z nejrozšířenějších a nejpoužívanějších programovacích jazyků. [23]

Oproti C++ je zde zakázaná vícenásobná dědičnost, která zaručuje odstranění diamantového efektu. Dále je zde přidán GC, který se stará o automatickou správu paměti. [23] Často lze zaslechnout přirovnání, že C# je Java od společnosti Microsoft. Svým způsobem toto přirovnání může být pravdivé, jelikož oba programovací jazyky jsou si velmi podobné. Markantním rozdílem jsou převážně jmenné konvencí a knihovní funkce. Zpravidla platí, že programátoři, kteří pracují s Javou, jsou schopni být velmi rychle přeškoleni na C# a opačně. Na obrázku 4 je zobrazen zdrojový kód napsaný v programovacím jazyku C#.

```
38          var response = new Response
39          {
40              Status = httpResponse.StatusCode,
41              Body = new Content()
42              {
43                  Value = await httpResponse.Content.ReadAsStringAsync(),
44              },
45
46          };
47
48
49          foreach (var (key :string, values :IEnumerable<string>) in httpResponse.Headers)
50          {
51              foreach (var value :string in values)
52              {
53                  if (key.ToLower() == SetCookie)
54                  {
55                      response.Cookies.Add(value);
56                  }
57                  else
58                  {
59                      response.Headers.Add((key, value));
60                  }
61
62              }
63          }
```

Obrázek 4 – Ukázka C#

1.4.6 WPF

Windows Presentation Foundation je knihovna pro vytváření uživatelských rozhraní za pomocí XAML, která je součástí .NET platformy. Hlavní výhodou je, že renderování probíhá na základě vektorových dat, tudíž je teoreticky možné zobrazit dané rozhraní na jakémkoliv rozlišení. Hlavní nevýhodou této technologie je podpora jen na operačním systému Windows, i když je kompletně převedená na .NET 6. Obdobně tomu je tak i na starší technologii Windows Forms. [24] Na

obrázku 5 je zobrazen zdrojový kód WPF prvku. Ve vývoji je nový framework MAUI, který již bude podporován na více platformách, nicméně dosud ještě nebyl vydán. [25]

Práce s daty v rozhraní může probíhat manuálně na základě zpracování dat z vytvořených komponent pomocí identifikátoru v daných metodách. Nicméně tento přístup se však nedoporučuje, jelikož je vhodné využít metody Binding za pomocí návrhového vzoru MVVM. [24]

```
<Grid Grid.Column="0"  >
    <ListView ItemsSource="{Binding Path=(???).Headers}"
              SelectedItem="{Binding Path=(???).SelectedHeader}"
              Margin="[l:]7,[t:]38,[r:]6,[b:]8" >
        <ListView.View>
            <GridView>
                <GridViewColumn Header="Name" Width="100">
                    <GridViewColumn.CellTemplate>
                        <DataTemplate>
                            <TextBox Text="{Binding Path=(???).Name}" TextWrapping="Wrap" Margin="[all:]0" Width="80"/>
                        </DataTemplate>
                    </GridViewColumn.CellTemplate>
                </GridViewColumn>
                <GridViewColumn Header="Value" Width="100">
                    <GridViewColumn.CellTemplate>
                        <DataTemplate>
                            <TextBox Text="{Binding Path=(???).Value}" TextWrapping="Wrap" Margin="[all:]0" Width="80"/>
                        </DataTemplate>
                    </GridViewColumn.CellTemplate>
                </GridViewColumn>

            </GridView>
        </ListView.View>
```

Obrázek 5 – Ukázka WPF v XAML

1.4.7 MVVM

Jedná se o návrhový vzor, který je rozdělen do třech komponentů, kterými jsou Model, View a ViewModel. Tento návrhový vzor odděluje uživatelské rozhraní od business logiky ve zdrojových kódech. Toto oddělení zvyšuje možnosti znovupoužití, jednodušší testování a celkovou flexibilitu. Hlavní výhodou je to, že umožní paralelizaci práce mezi vývojářem a designérem na daném UI. [26]

Pohled (anglicky view) je zdrojový kód čistě vizuálního uživatelského rozhraní, které může být implementováno různými způsoby XAML, HTML a jiné. Pod modelem je možné si představit data, se kterými aplikace pracuje. Pro dodání dat do UI je využit ViewModel, který funguje jako proxy mezi UI a business logikou. Data se z modelu předají do ViewModelu a daný ViewModel je poté vložen do View, který je zobrazí většinou přes tzv. Binding. Pro lepší představu byl vytvořen obrázek 6, který graficky zobrazuje diagram MVVM.

Obrázek 6 – MVVM diagram

1.4.8 Service Oriented Architecture

Service oriented architecture, česky architektura orientovaná na služby, je návrhový vzor architektury systému, který je rozdělen na mnoho jednoduchých služeb. Jednotlivé služby jsou často jednoúčelové a velmi efektivní. Pokud je SOA aplikované v jedné aplikaci, lze narazit na označení architektura mikroslužeb. Výhodou SOA je zjednodušení testování a velmi jednoduchá modifikace jednotlivých služeb. [27]

Každá služba musí mít nezávislý životní cyklus vzhledem k ostatním službám. Při vývoji jakékoliv služby by měl být kladen důraz na znovupoužitelnost v jiném systému. Správně navržené služby by měly splňovat podmínku bezestavovosti, nicméně tato podmínka může být opomenuta, pokud se jedná například o uložiště dat, kde je zapamatování stavu žádané. Dále by zde měl být praktikován princip abstrakce pro jednoduché modifikace nebo záměny implementací služeb. [28]

1.4.9 Dependency Injection

Vkládaní závislostí anglicky Dependency Injection je metoda využití IoC kontejneru. V principu se jedná o návrhový vzor, který vkládá závislosti do instancí třídy. Životní cykly daných instancí jsou spravovány třetí službou – IoC kontejnerem. Jednotlivé třídy jsou registrovány do kontejneru s vybraným typem životního cyklu. Samotné vkládaní závislostí lze rozdělit do dvou metod, resp. do tří. První metoda je vkládaní pomocí konstruktoru, další možností je vložení pomocí metody a poslední, která je spíše rozšíření druhé metody, pomocí Set metody, která je popsána rozhraním. [29] V praxi se zavolá IoC kontejner, který na vyžádání vrátí instanci dané třídy s již s vloženými závislostmi z jiných tříd.

Pomocí DI lze implementovat architekturu mikroslužeb, kdy jsou jednotlivé služby zaregistrovány s požadovaným typem (singleton, transient, scoped). Výhodou využití DI, jsou volné vazby mezi službami, nicméně toto řešení s sebou však nese i úskalí v podobě zvýšení režie, jelikož právě použití kontejneru zvyšuje složitost výpočtového času. Další problém vzniká při výskytu chyb, které nezachytí kompilátor a dostanou se tak až do runtime. [29]

2 NÁVRH A REALIZACE NÁSTROJE

Návrh aplikace zahrnuje analýzu, ve které lze nalézt funkční a nefunkční požadavky na vyvíjenou aplikaci. Dále tato kapitola zahrnuje popis vytvořených datových struktur. V podkapitole se čtenář dočte o návrhu a realizaci uživatelského rozhraní aplikace.

2.1 Požadavky

Sběr požadavků je jednou ze základních fází při vytváření analýzy pro vývoj softwaru. Hlavním úkolem tohoto sběru je získání přehledu o všech potřebných funkcionalitách a vlastnostech, které má daný software obsahovat. Samotné požadavky lze rozdělit do dvou hlavních kategorií. První kategorie jsou funkční požadavky, tyto požadavky definují přímé funkcionality a vlastnosti systému. Druhá kategorie jsou nefunkční požadavky, které specifikují další nepřímé požadavky, které se ale nevztahují samotnou funkcionalitu systému. Například se může jednat o nasazené technologie nebo požadavky na udržitelnost, bezpečnost, výkonnost apod.

2.1.1 Funkční požadavky

Cílem této práce je vytvoření nástroje pro automatické testovaní API s generátory požadavků včetně dat, proto byly vybrány následující funkční požadavky, které aplikace musí umožňovat:

- správa požadavků,
- odeslání požadavku na webový server,
- import specifikace OpenAPI,
- správa nastavení generátorů,
- generování požadavků z importované specifikace,
- vytvoření testu,
- vyhodnocení jednotlivého testu,
- hromadné vyhodnocení testů,
- uložení a načtení projektu.

Správa požadavků bude obsahovat možnost vytvoření, editaci a smazaní jednotlivých požadavků. Požadavek obsahuje adresu cíle, HTTP metodu, HTTP hlavičky a tělo dat. Každý správně definovaný požadavek lze odeslat na webový server, který vrací odpověď. Odpověď je zobrazena v aplikaci, aby ji uživatel mohl přehledně přečíst. Systém dále umožňuje nahrání OpenAPI

specifikace ze souboru nebo internetové stránky. Nahraná specifikace bude převedena do vnitřní struktury aplikace pro možnost rozšíření o jiné standardy. Danou strukturu si bude možné prohlédnout a nastavit u jednotlivých parametrů specifické hodnoty pro generátory, a to i včetně zasílaných těl požadavků. Aplikace dále umožní správu nastavení pro generátory, tato správa obsahuje vytvoření, modifikaci a smazání jednotlivých nastavení. Dále systém bude obsahovat možnost vytvoření testu, který je možné vyhodnotit. Dalším funkčním požadavkem je možnost hromadně vyhodnotit veškeré definované testy. Všechny vytvořené požadavky včetně nastavení generátorů a API specifikace bude možné uložit a zpětně načíst.

2.1.2 Nefunkční požadavky

Systém musí být napsaný ve frameworku .NET 6 za použití programovacího jazyka C#. Systém bude navržen na architektuře mikroslužeb. Jednotlivé služby jsou registrovány do IoC kontejneru za využití návrhového vzoru DI. Veškeré implementace služeb budou neveřejné a popsány veřejnými rozhraními. Dále projekt musí být rozdělen do dvou podprojektů, první z nich se bude starat o grafické rozhraní a druhý o samotnou funkcionalitu. Frontend (FE) bude obsahovat uživatelské rozhraní včetně pomocných tříd, které nemají žádnou spojitost s backendem. Backend (dále BE) bude obsahovat veškeré služby a modely byznysové logiky, které nemají spojitost s uživatelským rozhraním. FE bude moci využít veškeré veřejné třídy, rozhraní a výčty z BE, ale naopak však nikoliv. Výsledná aplikace bude spustitelná na operačním systému Windows 10 a vyšší.

2.2 Datové struktury

Tato podkapitola se zabývá popisem jednotlivých datových struktur, které byly vytvořeny pro uchovávání dat, se kterými aplikace pracuje na úrovni BE a částečně i FE.

2.2.1 Project

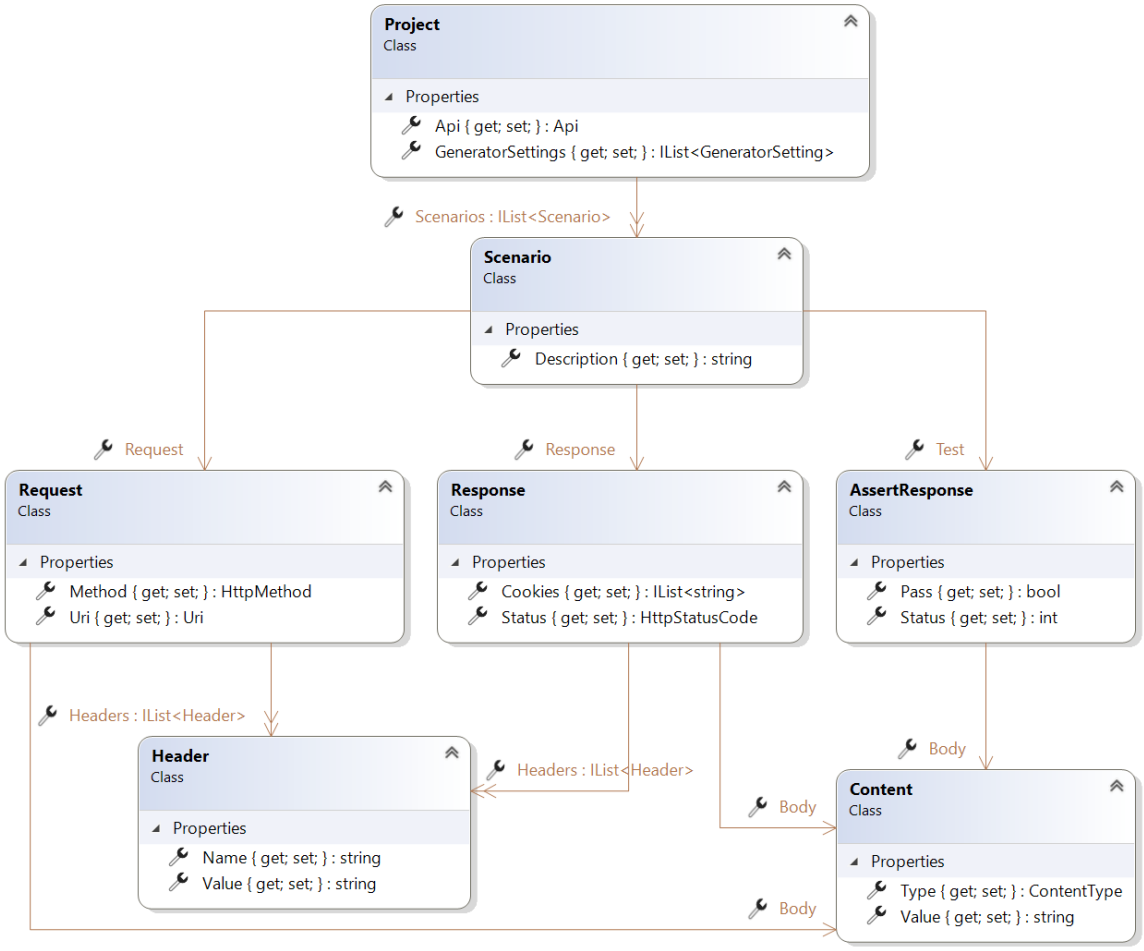
Třída Project a všechny třídy spjaté s touto třídou reprezentují nositele všech dat se kterými se uživatel setká při využívání aplikace. V samotném projektu se nachází kolekce scénářů, které seskupují požadavek, odpověď a test do jednoho logického celku. Dále se zde nachází kolekce nastavení generátorů a struktura importované specifikace API. Více informací o těchto dalších dvou strukturách se čtenář dozví v následujících podkapitolách.

Požadavek je tvořen HTTP metodou, URI adresou, tělem požadavku a hlavičkami. HTTP metoda je hodnota z výčtového typu, který obsahuje všechny metody definované ve standardu HTTP 1.1.

Tělo požadavku je nepovinná položka, která v sobě obsahuje hodnotu výčtového typu těla jakož je například JSON, HTML apod. a zároveň samotnou hodnotu těla.

Model odpovědi webového serveru je tvořen hlavičkami, cookies, stavovým kódem a tělem odpovědi. Zároveň je k tomuto modelu je přidružen model nazvaný AssertResponse, který reprezentuje očekávaná data, která jsou využita pro vyhodnocení automatického testu.

V následujícím obrázku 7 je zobrazena datová struktura modelů za pomocí UML diagramu, který přehledně vizualizuje výše popsané vlastnosti.

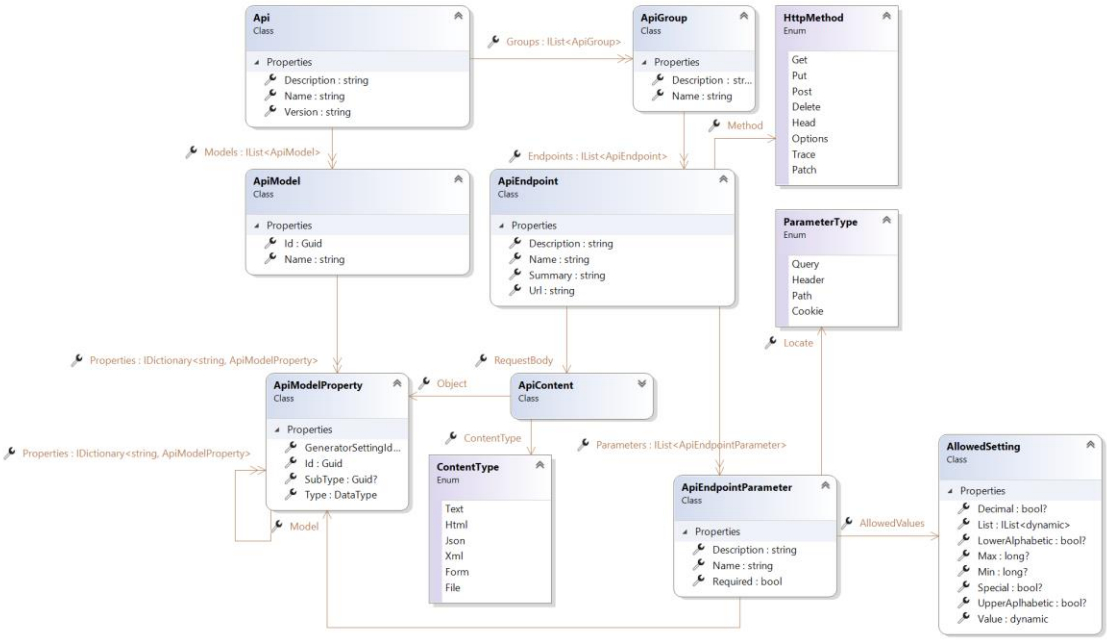


Obrázek 7 – UML Project

Při ukládání projektu je celá instance třídy Project serializována do formátu JSON a uložena do souboru. Zároveň při načítání dat je celá instance vytvořena pomocí deserializace ze souboru. Tuto funkci persistence zajišťuje služba ProjectPersistentService, která je popsána rozhraním IProjectPersistentService. Toto rozhraní obsahuje popis dvou asynchronních metod SaveProject a LoadProject.

2.2.2 API

Api model a všechny jeho složky jsou vytvořeny jako abstraktní struktura pro popis API. Po importování OpenAPI specifikace, dojde automaticky k převodu do této interní struktury a to za pomocí návrhového vzoru adaptér. Převedením dojde k vytvoření šablony pro generování testovacích požadavků. Následně je nutné tuto vytvořenou šablonu nakonfigurovat pomocí přiřazení jednotlivých nastavení generátorů do předem vytvořených parametrů. Pro lepší pochopení následného popisu byl vytvořen UML diagram, který vizualizuje danou strukturu, a je zobrazen na obrázku 8.



Obrázek 8 – UML API Model

Samotný kořenový model obsahuje jméno, popis a verzi v textových podobách, následuje kolekce třídy ApiModel, která reprezentuje znovupoužitelné modely v rámci jednotlivých šablon pro požadavky. Tato třída obsahuje jméno modelu, kolekci vlastností a unikátní id, které je využito pro volné vazby mezi modely a vlastnostmi. Zároveň kořenový model obsahuje kolekci skupin, které shlukují jednotlivé šablony požadavků pro lepší přehlednost.

ApiEndpoint je třída, kterou lze nazvat šablonou požadavků, jelikož obsahuje veškeré informace k vygenerování výsledného požadavku, který je umístěn ve struktuře scénáře. A to za předpokladu, že jsou korektně přiřazeny veškeré nastavení generátorů. Jednotlivé přiřazení mohou obsahovat parametry nebo tělo požadavku. Tělo požadavků nemusí být vytvořeno, jelikož existují případy, kdy požadavek nemusí přenášet přes tuto metodu žádná data. Dále tato třída obsahuje formátový typ cílového těla, kde je výsledná hodnota převedena do požadovaného typu, a vlastnost, která předepisuje datový typ pro generování hodnot.

Parametry mohou být typu cesty, dotazu, hlavičky nebo cookies. Zároveň obsahují jméno, popis a informaci o tom, zda je povinné daný parametr využít. Dále mohou obsahovat omezení u hodnot, například se může jednat o povolené výčtové typy nebo případně rozsah hodnot.

Třída ApiModelProperty výše popsána jako vlastnost je struktura, která popisuje datový typ pro generované hodnoty, případné Guid instance třídy ApiModel, pokud by se jednalo o datový typ Object. Dále obsahuje přiřazené nastavení generátorů. Detailnější informace o přiřazení tohoto nastavení jsou uvedeny v následující podkapitole zabývající se generováním dat.

Načtení OpenAPI specifikace obstarává služba OpenAPILoaderService, která obsahuje dvě asynchronní metody. Jedná se o načtení specifikace ze souboru a druhá načítá specifikaci z webové stránky. Obě tyto metody vrací strukturu OpenAPI modelů, která je distribuovaná s knihovnou OpenAPI.NET vytvořenou kolaborací společností Microsoft a OpenAPI Initiative.

2.3 Generování dat

Tato podkapitola se zabývá problematikou jednoho důležitého funkčního požadavku, kterým je generování testovacích scénářů z předem definovaných preferencí.

2.3.1 Generátory

Jednotlivé generátory jsou popsány rozhraním IGenerator. Jednotlivé implementace generátorů, které jsou popsány tímto rozhraním jsou zaregistrovány do ServiceCollection, který je využit pro dependency injection. Jelikož je nutné generovat data vícero datových typů byla vytvořena abstraktní generická třída GenericGenerator, která implementuje rozhraní IGenerator. Veškeré implementace generátorů jsou potomky této generické abstraktní třídy. Jednotlivé generátory lze rozdělit do třech kategorií, které jsou definovány výčtovým typem GeneratorType. Struktura generátorů je zobrazena na obrázku 9 pomocí UML diagramu.

Prvním nejjednodušším typem generátoru je generátor konstantní hodnoty. V pravém slova smyslu nejde o generátor nýbrž o poskytovatele konstantní hodnoty z nastavení generátoru, který je předán pomocí parametru metody. Tento typ byl zaveden, jelikož je nutné, aby uživatel měl možnost zadávat i konstantní hodnoty, protože v určitých případech je nutné mít předem definovanou hodnotu. Například se může jednat o velikost pole nebo zda daný objekt může mít hodnotu null. Samotná implementace toho typu generátoru je pouze jedna generická třída ConstantGenerator.

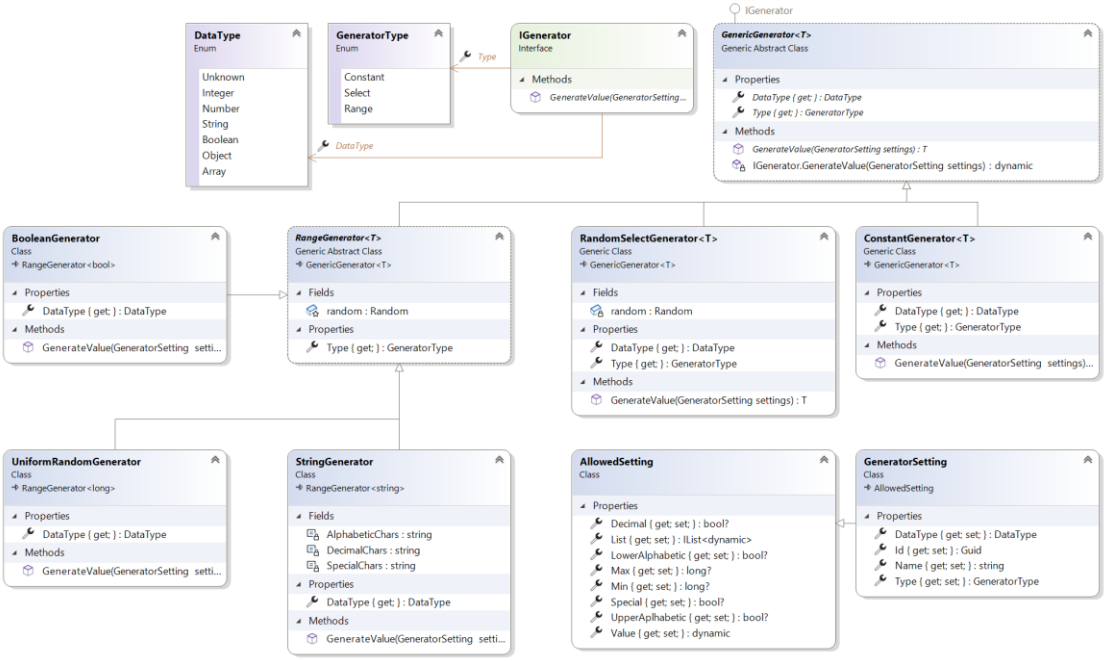
Dalším typem je náhodný výběr ze seznamu hodnot. Uživatel vytvoří nastavení, ve kterém nadefinuje hodnoty, ze kterých má daný generátor vybírat. Například se může jednat o jména nebo povolené hodnoty. Implementace je realizovaná pomocí generické třídy RandomSelectGenerator.

Posledním typem je generování rozsahu. Jedná o typ generátoru, který generuje data od minimální do maximální zadané hodnoty, nicméně tento popis nelze aplikovat na veškeré datové typy. V tomto případě bylo nutné, aby každý datový typ měl vlastní implementaci, jelikož výsledná data mohou být odlišná na druhu kontextu. Příkladem může být datový typ String nebo textový řetězec, který nelze popsat dvěma hodnotami. Nicméně pro dodržení přehlednosti kódu byla vytvořena abstraktní generická třída RangeGenerator ze kterých jsou dané implementace jejími potomky. Tato třída obsahuje atribut random, který všechny dědicí třídy využívají pro generování dat.

Generátor rozsahu pro datový typ Boolean generuje data true nebo false na základě vygenerovaného čísla a porovnání, zda vygenerované číslo je sudé, pokud je tato podmínka splněna, vrátí true v opačném případě vrací false. Implementace třídy je pojmenovaná BooleanGenerator.

UniforomRandomGenerator je generátor, který poskytuje vygenerovaná data typu Integer. Daný generátor využívá funkci rovnoměrného rozdělení náhodných čísel pro vygenerování čísla z nastaveného intervalu.

Rozsahový generátor pro datový typ textových řetězců je komplexnější nežli u ostatních datových typů. Samotný rozsah definuje velikost cílového řetězce a zároveň obsahuje možnosti nastavení povolení znaků. Jedná se o nastavení, které určuje možnosti použití abecedních, číselných a speciálních znaků. Zároveň jsou abecední znaky rozděleny na velké a malé znaky. Implementaci toho generátoru lze nalézt pod názvem StringGenerator.



Obrázek 9 – UML Generátorů

Generátory využívají generickou metodu GenerateValue, která obsahuje parametr typu GeneratorSetting. Tato generická metoda je volána v negenerické metodě abstraktní třídy GenericGenerator, která obsahuje návratový typ dynamic. GeneratorSetting je datový model, který obsahuje veškeré informace k parametrizaci jednotlivých generátorů.

2.3.2 Generování scénářů

Generování scénářů obstarává služba RequestGeneratorService, která obsahuje jedinou veřejnou metodu GenerateScenarios. Daná metoda přijímá v parametru celý projektový model a vrací vygenerované scénáře pomocí datové struktury IEnumerable. Metoda iteruje jednotlivé šablony požadavků obsažené ve skupinách. Ze samotného modelu šablony jsou převzaty adresy a HTTP metody, které jsou následně vloženy do nových instancí požadavků. Jednotlivé požadavky jsou vloženy do nových scénářů. Zároveň jsou v každé iteraci volány metody ResolveParameters a CreateRequestBody. Obě výše zmíněné metody využívají pro generování dat soukromou metodu GenerateData.

Metoda ResolveParameters iteruje parametry obsažené v šabloně požadavku, kde jsou data vygenerována a vložena podle jednotlivých typů do daného požadavku. Typ cesty upravuje adresu požadavku na základě změny vyznačeného jména parametru obklopeného složenými závorkami. Parametr typu Query přidává k adrese parametr v dotazovém formátu. Dále zde existují typy hlavička a cookies, které jednotlivé parametry v požadovaném formátu vkládají do hlaviček.

Metoda CreateRequestBody vrací tělo požadavku za podmínky, že jej šablona předepisuje. Pokud podmínka nebyla naplněna, vrací nulovou hodnotu. Při volání metody je výsledný model uložen do těla požadavku. Vygenerovaná data jsou převedena do formátu, který je popsán šablonou, a vložena do těla požadavku v požadovaným formátem.

GenerateData je metoda, která vrací vygenerovaná data pomocí vhodného generátoru a nastavení. Rekurzivní model ApiModelProperty obsahuje volnou vazbu na model nastavení generátoru pomocí jeho id. Pro vyhledání správného generátoru a nastavení se stará metoda GetGenerator, který podle id nalezne požadované nastavení. Následně je využit datový typ a typ generátoru, které jsou uloženy v nastavení, pro nalezení správného generátoru. Tento nalezený generátor je spolu s nastavením vrácen a později využit pro generování nových dat. GenerateData obsahuje přepínač podle datového typu uložený v modelu vlastnosti. Primitivní datové typy Integer, Number, String a Boolean jsou sjednoceny do jednoho bloku, který vrací vygenerovaná data. Pro další typy je řešení komplexnější. Pro typ pole je nejdříve vygenerovaný počet prvků, který se má generovat a poté jsou vygenerovány jednotlivé prvky pomocí rekurzivní metody GenerateData. Typ objekt nejdříve

generuje hodnotu typu boolean, která určuje, zda daný prvek má vygenerovat a pokud ano, tak vygeneruje daný objekt. Model vlastností obsahuje datovou strukturu typu tabulka, kde klíč udává jméno atributu daného objektu a hodnota je model typu ApiModelProperty. Proto je zde zvolen rekurzivní přístup pro vygenerování jednotlivých položek výsledného objektu.

2.4 Odesílání požadavků a vyhodnocování testů

Tato podkapitole se věnuje samotnému odesílání požadavků na server a následné vyhodnocování testů.

2.4.1 Odesílání požadavků

Pro odesílání požadavků na webový server byla vytvořena služba RequestSendService, která obsahuje asynchronní metodu Send. Tato metoda přijímá v parametru interní datový model Request a vrací odpověď z webového serveru. Samotná služba obsahuje soukromý atribut typu HttpClient, který je určen pro komunikaci s webovým serverem. Metoda Send vytváří instanci objektu HttpRequestMessage z poskytnutých dat modelu. Následně je zavolána metoda SendAsync webového klienta s parametrem výše zmíněné instance zprávy. Odpověď ze serveru je převedena do vnitřního modelu Response aplikace.

2.4.2 Testování požadavků

Vyhodnocování scénářů dochází v metodě Process, která je součástí služby TestRunnerService. Tato metoda využívá službu RequestSendService, díky které je zaslán požadavek na webový server a vrácena odpověď. Tato odpověď je následně zpracována výše zmíněnou metodou. Za předpokladu, že daný scénář obsahuje testovací model, začne být test vyhodnocován. Vyhodnocování probíhá na základě porovnání odpovědi a testovacího modelu Pokud jsou všechny data validní je prohlášen test za splněný. Není-li dané vyhodnocování validní, test je prohlášen za nevalidní.

2.5 Uživatelské rozhrání

Tato podkapitola popisuje problematiku uživatelského rozhraní a spjatých služeb a tříd. Uživatelské rozhraní je velmi důležitá část aplikace, jelikož je to prostředí pro přímý vstup a výstup mezi funkcionalitou aplikace a uživatelem. Samotné rozhraní by mělo být uživatelsky přívětivé, přehledné, intuitivní a snadné na ovládání. Pro tuto problematiku vznikla samostatná pracovní pozice UX designer, která řeší právě výše zmíněné aspekty. Cílem této pozice je vytvořit koncept aplikace, co se týče rozložení jednotlivých prvků a funkcionalit. Poté, co je navržen koncept, který může být plně implementován programátorem, nastupuje UI designer, který styluje jednotlivé prvky rozhraní do požadované podoby.

2.5.1 Služba pro správu datového modelu Project

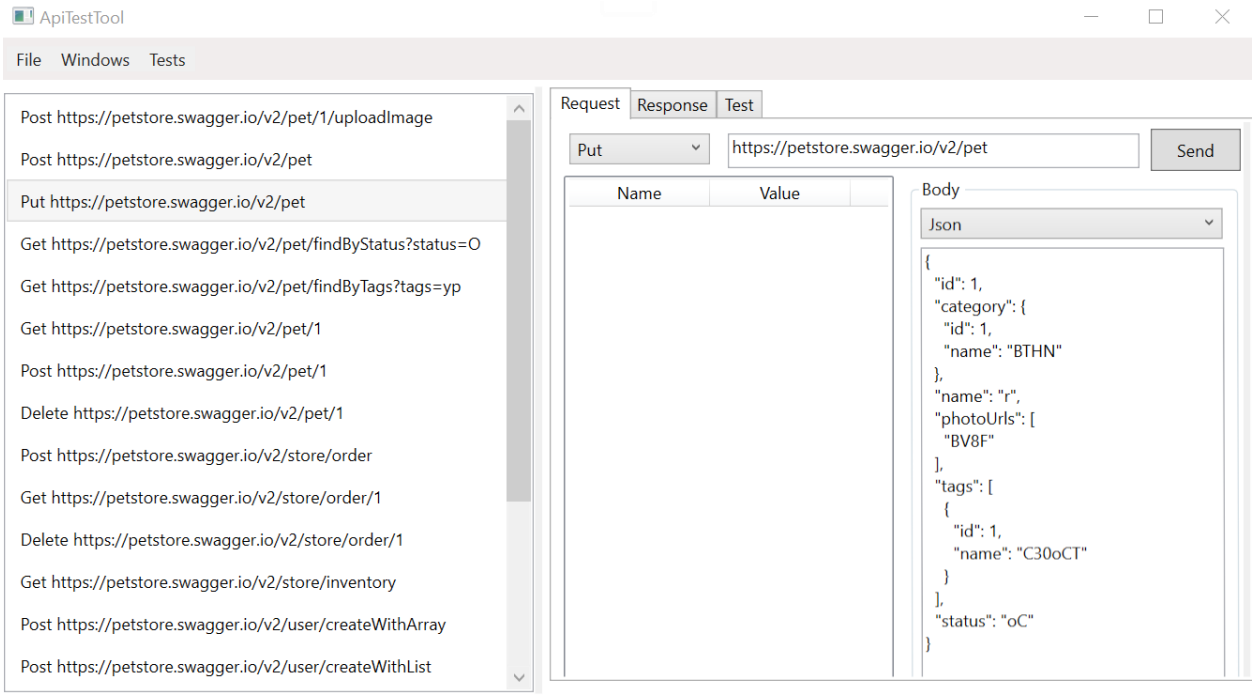
Pro správu datového modelu Project byla na FE vytvořena služba ProjectHandlerService, kde její instance je typu singleton. Cílem této služby je správa datového modelu Project pro dosažení synchronizace mezi jednotlivými okny. Tato služba obsahuje veřejnou vlastnost Project. Dále obsahuje metody pro uložení, načtení a založení nového projektu. Zároveň obsahuje metodu GetCountOfUsageSetting, která vrací počet použití daného nastavení podle jeho id.

2.5.2 Hlavní okno aplikace

Hlavní okno aplikace, jak již název napovídá je hlavní okno, které je zobrazeno při spuštění aplikace. Okno obsahuje menu, ve kterém je možné založit nový projekt, uložit jej, nebo načíst ze souboru. Dále menu obsahuje skupinu Windows, kterým lze zobrazit další dvě okna, kterým jsou věnovány následující podkapitoly. Zároveň obsahuje skupinu Tests, která je věnována spouštění automatických testů. Po levé straně okna se nachází list, ve kterém jsou zobrazeny veškeré vytvořené scénáře. Tento list obsahuje kontextové menu, které je určeno pro správu scénářů a nacházejí se zde položky pro vytvoření nového scénáře, smazání vybraného scénáře a jeho duplikaci. Při výběru z listu je zobrazen detail scénáře na pravé straně tohoto okna. Samotné rozložení je vizualizováno na obrázku 10.

Toto okno je složeno z pohledu MainWindow a jeho přidruženému viewmodelu MainWindowsViewModel. Daný model obsahuje kolekci scénářů a právě vybraný scénář. Samotná třída pohledu je rozdělena na dvě části, samotný pohled a část pro naprogramované události a další potřebné metody. Daná třída obsahuje metodu Refresh, která přebírá uložené scénáře z RequestSendService a následně jsou převedeny na viewmodely scénářů, které jsou vloženy do nové instance MainWindowsViewModel. Tato instance je poté vložena do pohledu pro využití

metody bindingu. Pří ukládání projektu jsou jednotlivé scénáře převedeny na modely a vloženy do služby, která spravuje Projekt a následně je zavolána metoda, která daný projekt uloží do souboru.



Obrázek 10 – Hlavní okno aplikace

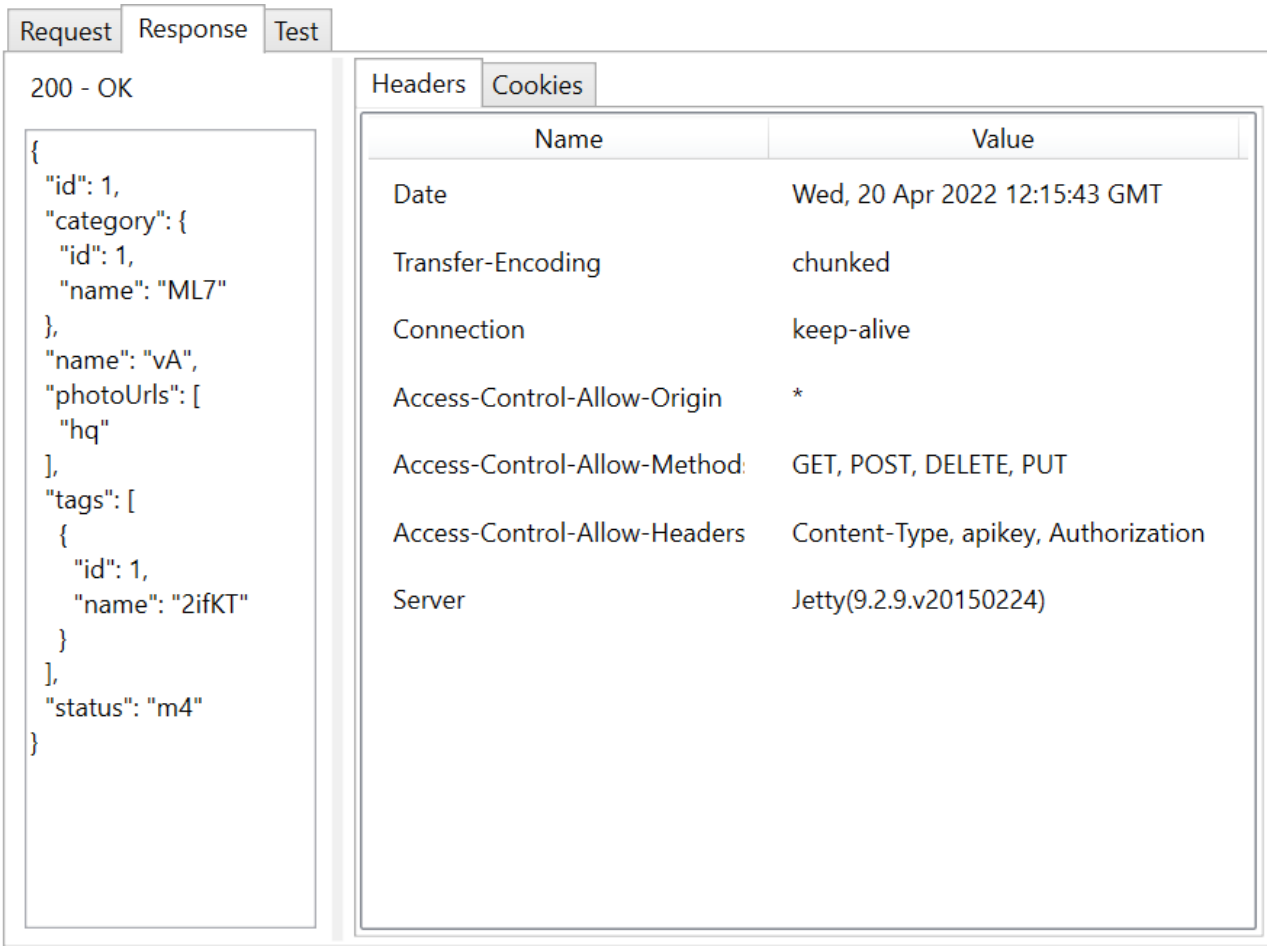
Při využití položky „Test All Scenarios“ v menu umístěném v horní části okna je zavolána událost, která prochází scénáře obsažené v projektu a pokud obsahují vytvořený test začne být vyhodnocován. Po vyhodnocení všech scénářů je zobrazeno dialogové okno, které obsahuje základní statistiku o počtu scénářů, počtu vyhodnotitelných testů a počet splněných testů.

Pro zpřehlednění vývoje byl detail scénáře vytvořen jako samostatný vizuální prvek typu Control pod třídou ScenarioControl. Samotný prvek obsahuje TabControl, který je určen jako přepínač jednotlivých záložek. Tento prvek obsahuje záložky Request, Response a Test. Každá záložka představuje jeden z viewmodelů, který je obsažen ve viewmodelu scénáře. Pro oddělení logiky má každý tento viewmodel vlastní vizuální prvek.

Vizuální prvek pro zobrazení dat požadavků se jmenuje RequestControl. Přidružený viewmodel se jmenuje RequestViewModel. V celém projektu jsou dodržovány tyto jmenné konvence, existuje-li datový model, přidružením slova Control nebo Window vzniká pohled typu prvku nebo okna. Pro změnu viewmodel je složen z jména modelu a přidaným slovem ViewModel. Tento prvek pro úpravu požadavku obsahuje v horní části combobox pro zvolení metody, textové pole pro adresu a tlačítko pro odeslání požadavku. Dále po levé straně se nachází ListView, který je určen pro hlavičky požadavků. Tento list obsahuje kontextové menu pro správu hlaviček, kde jednotlivé události jsou zpracovávány v třídě tohoto prvku. Po pravé straně se nachází seskupení

comboboxu pro typ těla požadavku a textového pole těla požadavku. Naplnění jednotlivých dat comboboxů je řešené v konstruktoru daného prvku. Událost tlačítka pro odeslání požadavku je nabindované ve viewmodelu.

Odpověď z webového serveru je zobrazována v ResponseControl. Uživatel při přepnutí na záložku „Response“ v detailu scénáře uvidí daný vizuální prvek. V levém horním rohu se nachází label, který zobrazuje stavový kód. Pod tímto labelem se nachází textové pole, ve kterém se nachází samotné tělo odpovědi v textové formě. Po pravé straně se nachází záložkový přepínač mezi hlavičkami a cookies. Obrázek 11 zobrazuje tento vizuální prvek.



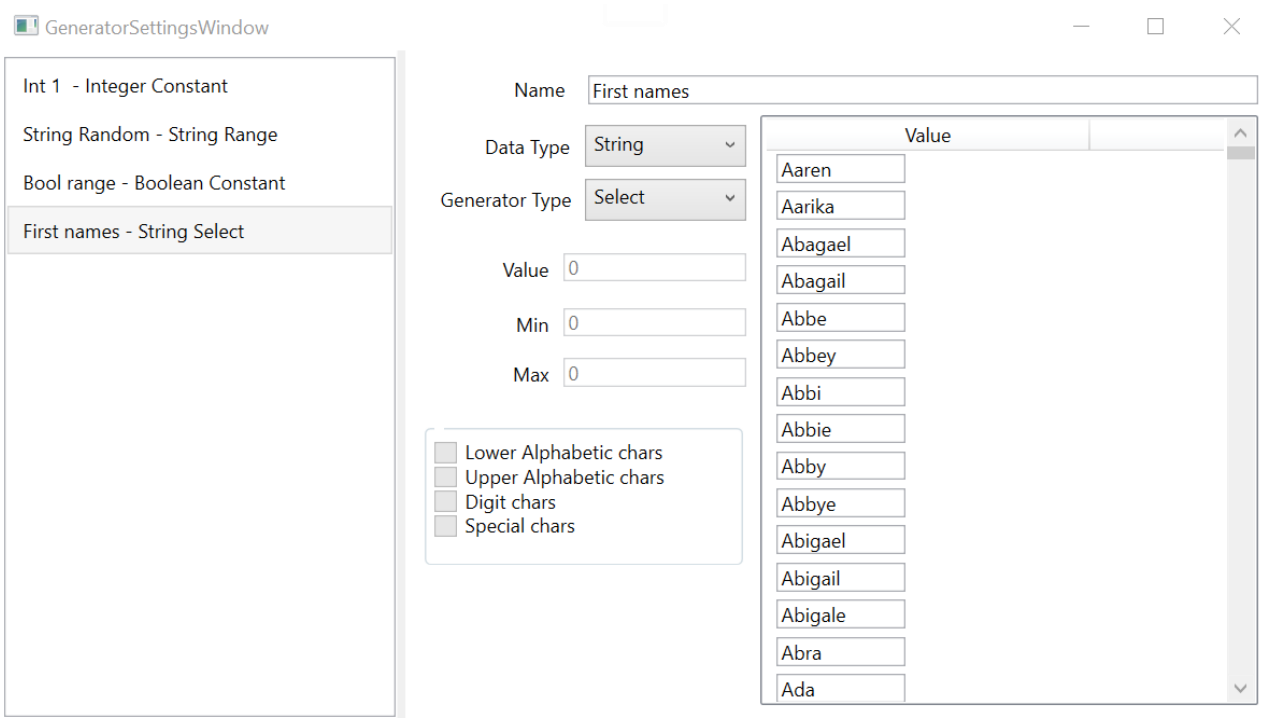
Obrázek 11 – Vizuální prvek odpovědi

Pod záložkou „Test“ se nachází vizuální prvek pro tvorbu automatizovaných testů. V horní části se nachází textové pole, které reprezentuje číslo stavového kódu odpovědi. Dále se zde nachází tlačítko pro spuštění testu a label, který obsahuje výsledek testu, zda byl splněn či nikoli. Ve spodní části se nachází textové pole pro vložení očekávané odpovědi s přepínačem pro formát dané odpovědi.

2.5.3 Správa nastavení generátorů

Okno pro správu nastavení generátorů je složeno z listu, ve kterém jsou vypsány jednotlivé položky nastavení a detailu vybraného nastavení. Daný list obsahuje kontextové menu, které dovoluje přidávat novou položku či její odstranění. Samotné okno je zobrazeno na obrázku 12.

Pro zpřehlednění a znovupoužitelnost byl detail nastavení vytvořen jako samostatný vizuální prvek GeneratorSettingControl, který je svázán se svým viewmodelem GeneratorSettingViewModel. Detail obsahuje textové pole jména nastavení, který slouží pro zpřehlednění dosazování a správu jednotlivých nastavení. Následuje přepínač pro nastavení datového typu, který je aktivní v případě, že není přiřazen v žádné vlastnosti šablony API. Následuje přepínač typu generátoru, který zpřístupňuje další položky nastavení.



Obrázek 12 – Správa nastavení generátorů

Pokud je vybrán konstantní typ generátoru je zpřístupněné pole textové pole hodnoty, které při zápisu kontroluje zadaný formát hodnoty. Jestliže formát hodnoty není přípustný nedojde k zápisu.

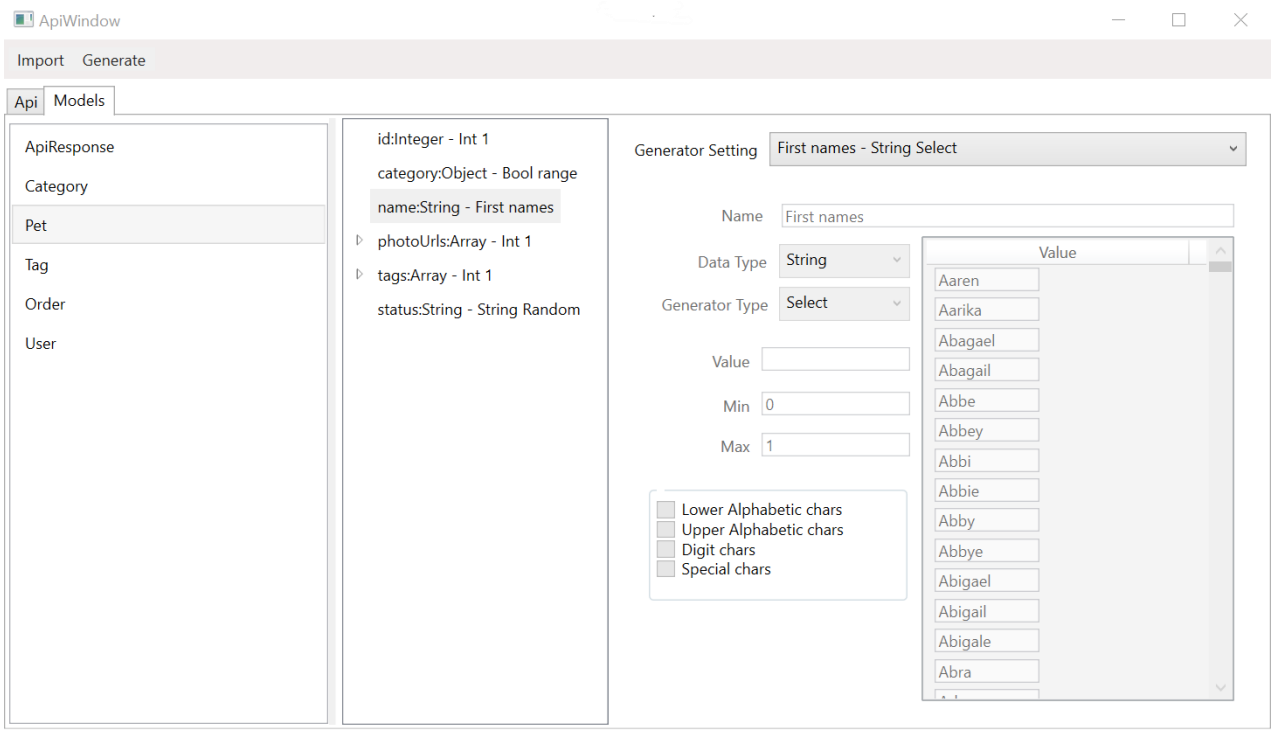
Za předpokladu vybraného typu generátoru výběru, je zpřístupněn list, který obsahuje seznam dat, které jsou vybírány náhodně vybírány generátorem. Daný list obsahuje kontextové menu pro správu těchto dat. Uživatel má možnost přidat novou položku, odstranit vybranou, vymazat všechny nebo načíst položky ze souboru. Načítání souboru je realizované pomocí textového načítání, kde každý nový řádek reprezentuje novou hodnotu. Jednotlivé hodnoty jsou kontrolovány, zda splňují požadovaný formát pro načtení datového typu. Jestliže soubor neobsahuje žádná validní data je uživateli předána zpráva o neplaných datech ve formátu dialogového okna.

Poslední typ generátoru je generátor rozsahu, kdy se zpřístupňují textové pole Min a Max. Textové pole Min reprezentuje minimální hodnotu generovaného rozsahu a textové pole Max zase maximální hodnotu. Pokud je zároveň vybraný datový typ String, jsou zpřístupněné položky typu checkbox, které slouží pro další nastavení generátoru. Jedná se o povolené znaky, které chce uživatel generovat v rámci daného generátoru. Zároveň pokud je vybrán datový typ Boolean jsou položky Min a Max znepřístupněny, jelikož tato hodnota má jen dvě validní hodnoty.

2.5.4 API

Okno API specifikace je vizuální reprezentací vnitřní struktury Api. Toto okno obsahuje v horní části menu, které obsahuje záložky Import a Generate. Pod záložkou Import, existuje položka „Import OpenAPI“. Událost po stisknutí této položky otevírá dialogové okno pro importování OpenAPI specifikace. Záložka „Generate“ obsahuje položku „All Requests“, jejíž událost má za úkol vygenerovat scénáře pomocí služby RequestGeneratorService. Zbytek toho okna je složen ze záložkového přepínače, který obsahuje samotnou API specifikaci a přidružené modely.

Záložka „API“ obsahuje po levé straně stromový seznam, který je strukturován hierarchicky. Samotné šablony požadavků jsou umístěny ve skupinách. Po pravé straně se nachází detail šablony požadavku. Pod záložkou „Models“ se nachází list modelů a detail právě vybraného modelu. Obrázek 13 zobrazuje samotné okno API specifikace.



Obrázek 13 – API okno

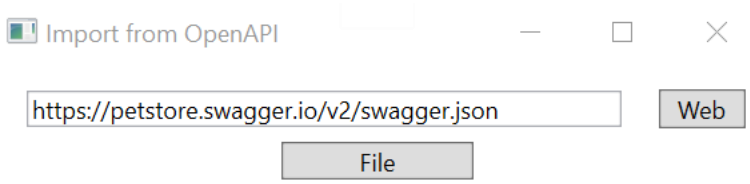
Detail šablony požadavků a detail modelu obsahuje totožný vizuální prvek ApiModelControl, který byl navržen pro jednoduchou znovupoužitelnost. Proto je možné využít daný prvek na obou záložkách. Po levé straně se nachází TreeView, který je využit pro zobrazení stromové struktury daných dat. Po pravé straně se nachází detail právě vybraného prvku z TreeView. V pravé horní části se nachází ComboBox, který obsahuje vytvořené nastavení generátorů kompatibilní s daným datovým typem. Zároveň tento seznam obsahuje i položku, která slouží pro zrušení vybraných nastavení. Za předpokladu, že je vybráno nastavení, jeho náhled je zobrazen pod daným výběrem,

pomocí vizuálního prvku GeneratorSettingControl. Tento vizuální prvek je určen jen pro čtení, proto nelze vnitřní data nijak upravovat. Pokud je ApiModelControl využíván v kontextu s modely API, jednotlivé položky ve stromovém listu reprezentují jednotlivé pole daného objektu. Za předpokladu, že je využíváno se spojením šablon požadavků, tak jednotlivé položky reprezentují jednotlivé parametry. Za předpokladu, že je vybrána položka typu Array, je potřeba využít generátor datového typu Integer, jelikož generátor vrátí počet položek, který je vygenerován. Při výběru datového typu Object, je nutné využít generátory datového typu Boolean, jelikož vygenerovaná hodnota z generátoru určuje, zda se daný objekt má generovat nebo je vrácena hodnota null.

2.5.5 Import OpenAPI specifikace

Dialogové okno „Import from OpenAPI“ je určeno pro importování OpenAPI specifikace ze souboru nebo z webového serveru. Dané okno obsahuje textové pole pro vložení URL adresy souboru. Vedle textového pole se nachází tlačítko „Web“, které slouží k načtení specifikace z webové stránky. Ve spodní části okna se nachází tlačítko „File“, které je určeno pro načtení specifikace ze souboru.

Toto dialogové okno využívá službu OpenAPILoaderService. Jednotlivé události zmáčknutí tlačítek používají tuto službu pro načtení, následně je datová struktura převedena na vnitřní strukturu aplikace pro specifikaci API. Převedená specifikace je vložena do veřejné vlastnosti, která je dále využitá pro distribuci do zbytku aplikace. Samotné dialogové okno je zobrazeno na obrázku 14.



Obrázek 14 – Import OpenAPI specifikace

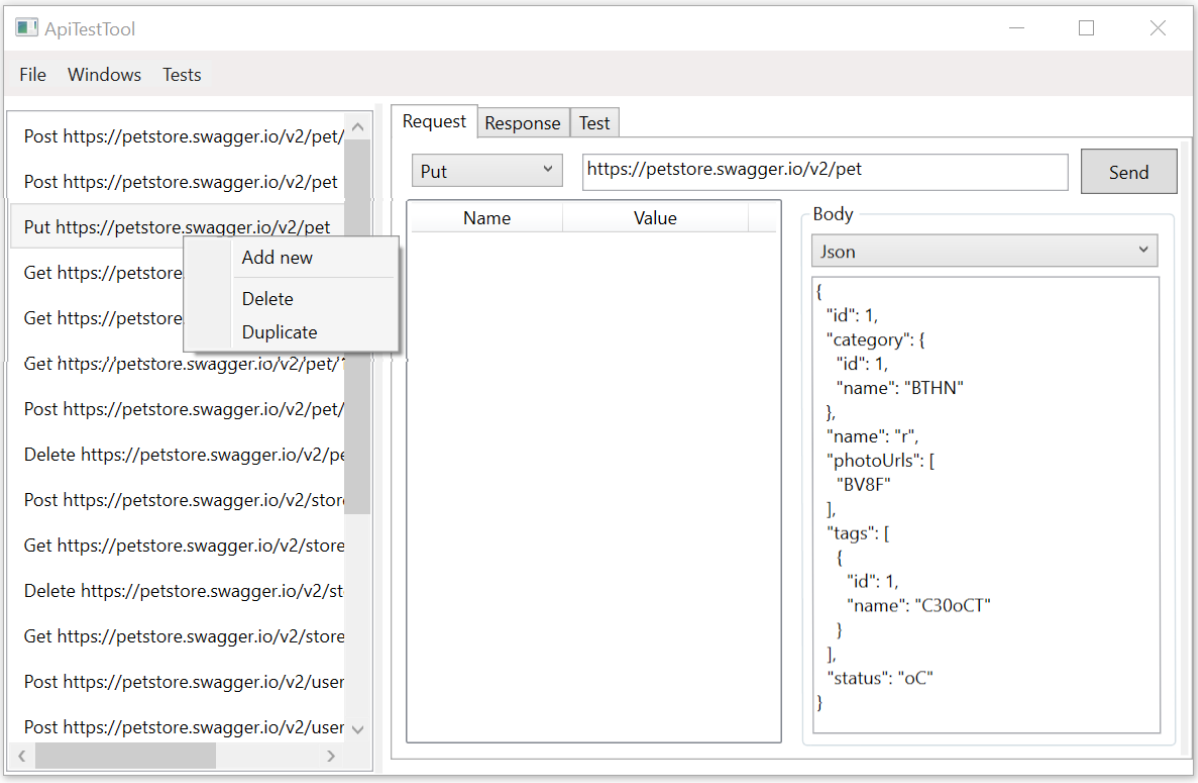
3 UŽIVATELSKÁ PŘÍRUČKA

Pro spuštění aplikace je nutné mít nainstalovaný .NET 6 desktop runtime knihovnu, kterou lze stáhnout z webových stránek společnosti Microsoftu. Při spuštění se zobrazí hlavní okno aplikace, která je spuštěna v režimu nového projektu. V následující podkapitole je popsané samotné rozložení hlavního okna včetně funkcionalit.

3.1 Hlavní okno aplikace

Hlavní okno aplikace obsahuje v horním pravém rohu menu, které je určené k provádění různých akcí, které jsou seskupeny do třech skupin. První skupina „File“ je věnována správě projektu, kde si uživatel může vytvořit, nahrát nebo uložit projekt. Následuje skupina „Windows“, která obsahuje položky pro zobrazení ostatních oken aplikace. Poslední skupina je zaměřena pro automatické testování, kde si uživatel může zapnout hromadné vyhodnocení testů.

V levé části okna se nachází seznam scénářů, který se při kliknutí pravého tlačítka myši na existujícím scénáři přepne na detail scénáře. Pokud uživatel klikne levým tlačítkem myši, zobrazí se kontextové menu, které slouží ke správě jednotlivých scénářů, které lze přidávat, odebírat nebo duplikovat. Na obrázku 15 je zobrazeno dané kontextové menu.



Obrázek 15 – Hlavní okno aplikace s vyplněnými daty

3.1.1 Vytvoření scénáře

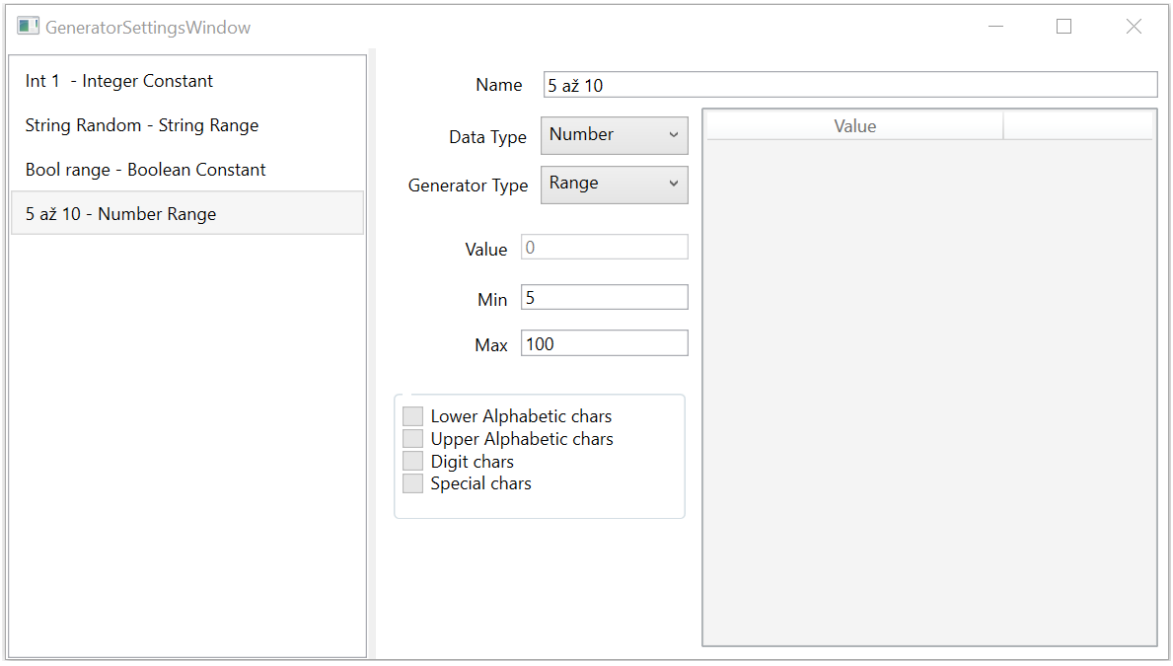
Nejprve je nutné pravým tlačítkem myši kliknout na seznam scénářů a vybrat položku pro vytvoření scénáře. Tímto krokem se vytvoří nový scénář, který je nutné nastavit v detailu scénáře. Nejdůležitější je nastavit požadovanou URL adresu, pro kterou slouží textové pole vedle tlačítka „Send“. Následně je nutné zvolit požadovanou metodu daného webového požadavku. Tato metoda lze nastavit pomocí přepínače, který se nachází vedle URL adresy. Po zvolení metody je možné vyplnit tělo dotazu s požadovaným typem těla. Dále je možné nastavit hlavičky pomocí levého tlačítka myši na listovém seznamu hlaviček. Po vyplnění všech potřebných hodnot je možné daný požadavek zaslat na server pomocí tlačítka „Send“. Odpověď z webového serveru se nachází pod záložkou „Response“.

3.1.2 Vytvoření automatického testu

V detailu scénáře se nachází záložka „Test“, která slouží pro vytváření automatických testů. Po kliknutí na tuto záložku je zobrazen detail, kde je možné vyplnit očekávaná data. Zároveň se zde nachází tlačítko „Test“, které daný test spustí a následně vyhodnotí.

3.2 Nastavení generátorů

Pro zobrazení okna nastavení generátorů je nejprve nutné v hlavním okně aplikace zakliknout položku „Generator settings“, která se nachází v menu pod hlavní záložkou „Windows“. Obrázek 16 zobrazuje okno nastavení generátorů s vyplněnými daty.



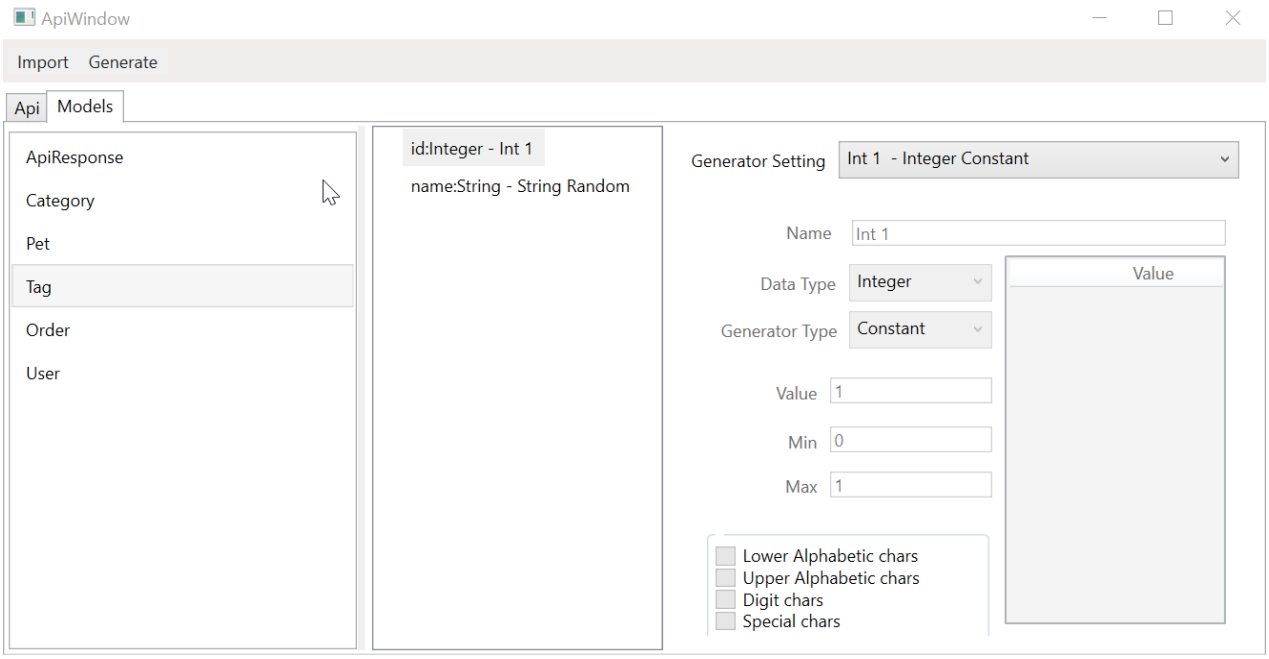
Obrázek 16 – Okno nastavení generátorů

3.2.1 Vytvoření nastavení pro generátor

Nejprve je nutné zobrazit si okno, které obsahuje správu daných nastavení. V levé části tohoto okna se nachází seznam, který obsahuje kontextové menu pro správu jednotlivých nastavení. Aktivace tohoto menu se provede prostřednictvím pravého tlačítka myši na tomto seznamu. Pro vytvoření nového nastavení je potřebné kliknout v tomto menu na položku „Add new“. Následně se nastavení vytvoří a lze jej konfigurovat v zobrazeném detailu.

3.3 Okno API

V hlavním okně aplikace je nutné vybrat položku „API Window“ pro zobrazení tohoto API okna. Okno API obsahuje v horní časti menu, které obsahuje položky pro importování specifikace OpenAPI a generování scénářů podle nahrané a nakonfigurované specifikace. Obrázek 17 zobrazuje vybranou vlastnost API modelu.



Obrázek 17 – API okno s daty

3.3.1 Import OpenAPI specifikace

Pro importování OpenAPI specifikace je nutné nejdříve zobrazit dialogové okno k tomu určené. Tuto akci lze provést prostřednictvím menu v API okně. Následně v dialogovém okně jsou dvě možnosti, jak nahrát specifikaci. První možnost je ze souboru, v dialogovém okně je nutné kliknout na tlačítko na tlačítko „File“. Při této akci se zobrazí nahrávací formulář, kde je nutné vybrat daný soubor. Druhá možnost je načtení z webové stránky. Pro tuto možnost je nutné vyplnit URL adresu do připraveného textového pole a následně kliknout na tlačítko „Web“.

ZÁVĚR

Bakalářská práce se věnuje návrhu a vývoji nástroje pro automatické testovaní REST API. Samotná práce je rozdělena na dvě poloviny.

V první polovině je čtenář seznámen se samotnou definicí testování, různými druhy testů a jejich aplikovatelností v praxi. Jsou zde zmíněny již existující řešení této problematiky a popsán jejich přístup. Následně se práce zabývá popisem technologií, které jsou využity v implementaci nástroje. Nejprve se věnuje webovým technologiím, přesněji HTTP, RESTful API a následně specifikaci OpenAPI. Následuje platforma .NET s programovacím jazykem C#. Nechybí ani popis knihovny WPF, která je určena pro vytváření uživatelského rozhraní. Dále je popsaný princip návrhového vzoru MVVM, který byl vytvořen přímo pro komunikaci s WPF a zbytkem zdrojového kódu. Nakonec je zmíněna architektura orientované na služby, související pojem mikroslužby popis reálné aplikovatelnosti mikroslužeb v rámci návrhového vzoru Dependency Injection za pomocí využití IoC kontejneru.

Druhá polovina práce se zabývá návrhem a implementací testovacího nástroje. V první podkapitole jsou definovány sesbírané funkční a nefunkční požadavky, podle kterých byla následně aplikace realizována. V následující podkapitole jsou popsány veškeré datové struktury, které jsou využity pro uchování uživatelských dat. Nechybí podkapitola, která je věnována samotnému generování dat, které je jedním z nejdůležitějších požadavků cílové aplikace. V této části je nejdříve popsán princip návrhu a funkčnosti vytvořených generátorů. Následně je popsána realizace generování scénářů za použití generátorů s nastavenými parametry. V předposlední části jsou popsány principy odesílání požadavků na webový server a následné testování odpovědí z webového serveru. V závěru je čtenáři představeno uživatelské rozhraní. Jsou zde popsána jednotlivá okna, jejich účel a dále také funkcionality spjaté s uživatelským rozhraním. Tato část zároveň slouží jako uživatelská příručka a pomáhá pochopit vytvořenou aplikaci a naučit se jejímu používání.

POUŽITÁ LITERATURA

[1] What is .NET? Introduction and overview. *Microsoft technical documentation* [online]. Redmond: Microsoft, © 2022, 03/11/2022 [cit. 2022-03-25]. Dostupné z: https://docs.microsoft.com/en-us/dotnet/core/introduction

[2] Postman API Platform [online]. San Francisco: Postdot Technologies, © 2022 [cit. 2022-03-20]. Dostupné z: https://www.postman.com

[3] Katalon: An all-in-one test automation solution [online]. Atlanta: katalon, © 2022 [cit. 2022-03-22]. Dostupné z: https://katalon.com/

[4] AMMANN, Paul a Jeff OFFUTT. Introduction to Software Testing [online]. 2nd ed. Cambridge (UK): Cambridge University Press, 2018 [cit. 2022-01-05]. ISBN 9781316771273. DOI:10.1017/9781316771273 Dostupné z: http://dx.doi.org/10.1017/9781316771273

[5] MASSÉ, Mark. REST API Design Rulebook: Designing Consistent RESTful Web Service Interfaces. Sebastopol(CA): O'Reilly, © 2012. ISBN 978-1-449-31050-9.

[6] BANIAȘ, Ovidiu, Diana FLOREA, Robert GYALAI a Daniel-Ioan CURIAC. Automated Specification-Based Testing of REST APIs [online]. Sensors, 2021, 21(16), 5375 [cit. 2022-01-05]. Dostupné z: doi:10.3390/s21165375

[7] BRABRA, Hayet, Achraf MTIBAA, Fabio PETRILLO, et al. On semantic detection of cloud API (anti)patterns. Information and Software Technology [online]. Elsevier B.V., March 2019, 107(2019), 65-82 [cit. 2022-01-05]. ISSN 0950-5849. Dostupné z: doi:10.1016/j.infsof.2018.10.012

[8] FEWSTER, Mark a Dorothy GRAHAM. Software Test Automation: Effective Use of Test Execution Tools. New York: ACM Press, 1999. ISBN 978-0201331400.

[9] FIELDING, Roy. Architectural Styles and the Design of Network-based Software Architectures. Irvine, 2000. Disertační práce. University of California.

[10] QUADRI, S.M.K a Sheikh Umar FAROOQ. Software Testing – Goals, Principles, and Limitations. International Journal of Computer Applications [online]. 2010, 6(9), 7-10 [cit. 2022-01-17]. ISSN 09758887. Dostupné z: doi:10.5120/1343-1448

[11] BOURQUE, Pierre a Richard FAIRLEY, ed. *SWEBOK: Guide to the Software Engineering Body of Knowledge*. 3.0. Los Alamitos, CA: IEEE Computer Society, 2014. ISBN 0-7695-5166-1.

[12] SoapUI: Accelerating API Quality Through Testing [online]. Somerville (Massachusetts): SmartBear Software, © 2021 [cit. 2022-03-22]. Dostupné z: https://www.soapui.org

[13] FIELDING, Roy a Julian RESCHKE, ed. *Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing* [online]. June 2014 [cit. 2022-03-22]. Dostupné z: https://datatracker.ietf.org/doc/html/rfc7230

[14] BELSHE, Mike a Roberto PEON, THOMSON, Martin, ed. *Hypertext Transfer Protocol Version 2 (HTTP/2)* [online]. May 2015 [cit. 2022-03-22]. Dostupné z: https://datatracker.ietf.org/doc/html/rfc7540

[15] SATRAPA, Pavel. *Jak funguje nový protokol HTTP/2. Root.cz* [online]. 4. 3. 2015 [cit. 2022-03-23]. Dostupné z: https://www.root.cz/clanky/jak-funguje-novy-protokol-http-2/

[16] BISHOP, Mike, ed. *Hypertext Transfer Protocol Version 3 (HTTP/3): draft-ietf-quic-http-34* [online]. 2 February 2021 [cit. 2022-03-23]. Dostupné z: https://datatracker.ietf.org/doc/html/draft-ietf-quic-http-34¨

[17] *Comparison of the usage statistics of HTTP/2 vs. HTTP/3 for websites* [online]. Q-Success, © 2009-2022 [cit. 2022-03-23]. Dostupné z: https://w3techs.com/technologies/comparison/ce-http2,ce-http3

[18] MILLER, Darrel, Jeremy WHITLOCK, Marsh GARDINER, Mike RALPHSON, Ron RATOVSKY, Jason HARMON a Tony TAM, ed. OpenAPI Specification v3.1.0. *OpenAPI Initiative* [online]. The Linux Foundation, 15 February 2021 [cit. 2022-03-24]. Dostupné z: https://spec.openapis.org/oas/v3.1.0

[19] WAGNER, Janet. What's the Difference Between OpenAPI 2.0, 3.0, and 3.1?. *Stoplight* [online]. Austin: Stoplight, September 29 2020 [cit. 2022-03-24]. Dostupné z: https://blog.stoplight.io/difference-between-open-v2-v3-v31

[20] *Swagger* [online]. Somerville (Massachusetts): SmartBear Software, © 2021 [cit. 2022-03-24]. Dostupné z: https://swagger.io

[21] LINNIK, Irina. A Brief History of .NET Framework. *SoftTeco* [online]. Vilnius: SoftTeco, © 2008-2021, 13 June 2019 [cit. 2022-03-25]. Dostupné z: https://softteco.com/blog/a-brief-history-of-net-framework

[22] .NET and .NET Core Support Policy. *Microsoft* [online]. Redmond: Microsoft, © 2022 [cit. 2022-03-26]. Dostupné z: https://dotnet.microsoft.com/en-us/platform/support/policy/dotnet-core

[23] A tour of the C# language. *Microsoft technical documentation* [online]. Redmond: Microsoft, © 2022, 03/18/2022 [cit. 2022-03-26]. Dostupné z: https://docs.microsoft.com/en-us/dotnet/csharp/tour-of-csharp/

[24] Desktop Guide (WPF .NET). *Microsoft technical documentation* [online]. Redmond: Microsoft, © 2022, 04/15/2021 [cit. 2022-03-26]. Dostupné z: https://docs.microsoft.com/en-us/dotnet/desktop/wpf/overview/?view=netdesktop-6.0

[25] ORTINAU, David. Announcing .NET MAUI Preview 11. *Microsoft Developer Blogs* [online]. Redmond: Microsoft, © 2022, January 5th, 2022 [cit. 2022-03-26]. Dostupné z: https://devblogs.microsoft.com/dotnet/announcing-dotnet-maui-preview-11/

[26] VISHNUJEET, Kumar. MVVM: Model-View-ViewModel. *C# Corner* [online]. C# Corner, © 2022, Feb 03, 2021 [cit. 2022-03-26]. Dostupné z: https://www.c-sharpcorner.com/UploadFile/0b73e1/mvvm-model-view-viewmodel-introduction-part-1/

[27] SOA (Service-Oriented Architecture): Explore SOA (service-oriented architecture), an important stage in the evolution of application development and integration. *IBM* [online]. New York: IBM, [2021], 7 April 2021 [cit. 2022-03-27]. Dostupné z: https://www.ibm.com/cloud/learn/soa

[28] HAAN, Johan. *Architecture requirements for Service-Oriented Business Applications* [online]. MAY 19, 2008 [cit. 2022-03-27]. Dostupné z: http://www.theenterprisearchitect.eu/blog/2008/05/19/architecture-requirements-for-service-oriented-business-applications/

[29] KARIA, Bhavya. *A quick intro to Dependency Injection: what it is, and when to use it* [online]. OCTOBER 18, 2018 [cit. 2022-03-27]. Dostupné z: https://www.freecodecamp.org/news/a-quick-intro-to-dependency-injection-what-it-is-and-when-to-use-it-7578c84fa88f/

[30] ROUDENSKÝ, Petr a Anna HAVLÍČKOVÁ. Řízení kvality softwaru: průvodce testováním. Brno: Computer Press, 2013. ISBN 978-80-251-3816-8.