UNIVERZITA PARDUBICE
Fakulta ekonomicko-správní
Ústav systémového inženýrství a informatiky

Integrační testování s využitím nástroje Postman

DIPLOMOVÁ PRÁCE

2023 Bc. STANISLAV HUŇÁČEK

Univerzita Pardubice
Fakulta ekonomicko-správní
Akademický rok: 2022/2023

ZADÁNÍ DIPLOMOVÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Bc. Stanislav Huňáček**
Osobní číslo:	**E21060**
Studijní program:	**N0688A140007 Informatika a systémové inženýrství**
Specializace:	**Informační a bezpečnostní systémy**
Téma práce:	**Integrační testování s využitím nástroje Postman**
Zadávající katedra:	**Ústav systémového inženýrství a informatiky**

Zásady pro vypracování

Cílem práce je vytvoření manuálu zabývajícího se API testováním s využitím nástroje Postman.
Osnova:
- Popis současného stavu (objasnění základních pojmů problematiky testování).
- Formulace problému.
- Integrační testování s využitím nástroje Postman (application programming interface, Postman, příklad testování a vytvoření manuálu).
- Zhodnocení dosažených výsledků.

Rozsah pracovní zprávy: **cca 50 stran**
Rozsah grafických prací:
Forma zpracování diplomové práce: **tištěná/elektronická**

Seznam doporučené literatury:

BUREŠ, Miroslav, Miroslav RENDA, Michal DOLEŽEL, Peter SVOBODA, Zdeněk GRÖSSL, Martin KOMÁREK, Ondřej MACEK a Radoslav MLYNÁŘ, 2016. Efektivní testování softwaru: klíčové otázky pro efektivitu testovacího procesu. Praha: Grada. Profesionál. ISBN 978-80-247-5594-6.
PATTON, Ron, 2002. Testování softwaru. Praha: Computer Press. Programování. ISBN 80-7226-636-5.
RICHARDSON, Alan, 2017. Automating and Testing a REST API: A Case Study in API testing using: Java, REST Assured, Postman, Tracks, cURL and HTTP Proxies. Compendium Developments. ISBN 978--0956733290.
STEPHENS, Matt a Doug ROSENBERG, 2011. Testování softwaru řízené návrhem. Brno: Computer Press. ISBN 978-80-251-3607-2.
WESTERVELD, Dave, 2021. API Testing and Development with Postman: A practical guide to creating, testing, and managing APIs for automated software testing. Birmingham, England: Packt Publishing. ISBN 978-1800569201.
WINTERINGHAM, Mark, 2022. Testing Web APIs. Manning. ISBN 978-1617299537.

Vedoucí diplomové práce: **Ing. Miloslava Kašparová, Ph.D.**
Ústav systémového inženýrství a informatiky

Datum zadání diplomové práce: **1. září 2022**
Termín odevzdání diplomové práce: **30. dubna 2023**

L.S.

prof. Ing. Jan Stejskal, Ph.D. v.r.
děkan

RNDr. Ing. Oldřich Horák, Ph.D. v.r.
vedoucí ústavu

V Pardubicích dne 1. září 2022

PROHLÁŠENÍ AUTORA

Tuto práci Integrační testování s využitím nástroje Postman 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., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, 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 od 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. 7/2019 Pravidla pro odevzdávání, zveřejňování a formální úpravu závěrečných prací, ve znění pozdějších dodatků, bude práce zveřejněna prostřednictvím Digitální knihovny Univerzity Pardubice.

V Pardubicích dne 30. 06. 2023.

Stanislav Huňáček v. r.

PODĚKOVÁNÍ

Tímto bych rád poděkoval své vedoucí práce Ing. Miloslavě Kašparové, Ph.D. za její odbornou pomoc, doporučení a cenné rady, které mi pomohly při zpracování diplomové práce. Dále bych chtěl vyjádřit poděkování své rodině, přítelkyni a přátelům, kteří mě při tvorbě diplomové práce neustále podporovali.

ANOTACE

Hlavním cílem této diplomové práce je vytvoření manuálu, který bude sloužit jako příručka k testování API softwarovým nástrojem Postman. V rámci práce bude provedeno integrační testování na veřejně dostupném API s využitím nástroje Postman. Diplomová práce se dále zaměřuje na využití základních funkcí nástroje Postman pro integrační testování API a obsahuje popis současného stavu problematiky testování, přičemž objasňuje základní pojmy v této oblasti.

KLÍČOVÁ SLOVA

API, integrační, manuál, Postman, testování

TITLE

Integration testing using the Postman tool

ANNOTATION

The primary objective of this thesis is to develop a comprehensive manual that will act as a practical guide for conducting API testing using the Postman software tool. As a component of the thesis, the Postman tool will be utilized to perform integration testing on a publicly accessible API. Additionally, the thesis will concentrate on the fundamental features of the Postman tool for API integration testing and provide an overview of the existing testing practices, while elucidating the basic concepts in this domain.

KEYWORDS

API, integration, manual, Postman, testing

OBSAH

SEZNAM ILUSTRACÍ A TABULEK **9**

SEZNAM ZKRATEK A ZNAČEK **10**

ÚVOD **11**

1 POPIS SOUČASNÉHO STAVU TESTOVÁNÍ SOFTWARU **12**

1.1 HISTORIE TESTOVÁNÍ SOFTWARU 12

1.1.1 Období ladění 12

1.1.2 Demonstrační období 12

1.1.3 Destruktivní období 13

1.1.4 Období orientované na hodnocení 13

1.1.5 Období zaměřené na prevenci 13

1.2 STATICKÉ TESTOVACÍ TECHNIKY 13

1.2.1 Revize 14

1.2.2 Statická analýza 15

1.3 DYNAMICKÉ TESTOVACÍ TECHNIKY 16

1.3.1 Black-box testování 16

1.3.2 White-box testování 17

1.3.3 Testování založené na zkušenostech testera 17

1.4 TESTOVÁNÍ V RÁMCI ŽIVOTNÍHO CYKLU VÝVOJE SOFTWARU 17

1.4.1 Vodopádový model 18

1.4.2 Spirálový model 19

1.4.3 V-model a W-model 21

1.4.4 Rational Unified Process model 22

1.4.5 Scrum model 23

1.4.6 Rapid application development model 24

1.5 ÚROVNĚ TESTOVÁNÍ 26

1.5.1 Testování jednotek 27

1.5.2 Integrační testování 29

1.5.3 Systémové testování 33

1.5.4 Akceptační testování 34

2 FORMULACE PROBLÉMU **35**

3 INTEGRAČNÍ TESTOVÁNÍ S VYUŽITÍM NÁSTROJE POSTMAN 36

3.1 HTTP PROTOKOL 36

3.1.1 Metody HTTP požadavků 36

3.1.2 HTTP stavové kódy 37

3.2 API 38

3.2.1 REST API 38

3.2.2 SOAP API 39

3.2.3 GraphQL API 40

3.2.4 gRPC API 41

3.3 POSTMAN 42

3.3.1 API klient 42

3.3.2 Tarify aplikace Postman 44

3.3.3 Další nástroje pro testování API 45

3.4 PŘÍKLAD NA TESTOVÁNÍ API 47

3.4.1 Popis zvoleného API 47

3.4.2 Návrh testovacích případů 49

3.4.3 Implementace testovacích případů 50

3.4.4 Výsledky exekuce testovacích případů 52

3.5 VYTVOŘENÍ MANUÁLU K APLIKACI POSTMAN 54

4 ZHODNOCENÍ DOSAŽENÝCH VÝSLEDKŮ 56

ZÁVĚR 57

POUŽITÁ LITERATURA 58

PŘÍLOHY 64

PŘÍLOHA A – NAVRŽENÉ TESTOVACÍ PŘÍPADY 65

PŘÍLOHA B – IMPLEMENTOVANÉ TESTOVACÍ PŘÍPADY 71

PŘÍLOHA C – NÁVOD K APLIKACI POSTMAN 82

SEZNAM ILUSTRACÍ A TABULEK

Obrázek 1 – Relativní náklady na opravu chyby dle času odhalení 14
Obrázek 2 – Black-box testování 16
Obrázek 3 – Schéma vodopádového modelu 19
Obrázek 4 – Spirálový model 20
Obrázek 5 – V-model 21
Obrázek 6 – W-model 22
Obrázek 7 – Rational Unified Process model 23
Obrázek 8 – Scrum model 24
Obrázek 9 – Rapid application development model 25
Obrázek 10 – Úrovně testování 26
Obrázek 11 – Přístupy k integračnímu testování 31
Obrázek 12 – Bottom-up přístup 31
Obrázek 13 – Top-down přístup 32
Obrázek 14 – Průběh systémového testování 33
Obrázek 15 – REST API model 39
Obrázek 16 – Struktura SOAP zprávy 40
Obrázek 17 – Organizace do složek 50
Obrázek 18 – Nastavené parametry v Collection Runneru 53
Obrázek 19 – Výsledky exekuce 54
Obrázek 20 – Detail nalezené chyby - getDistrictListAsJson 54

Tabulka 1 – Porovnání úrovní testování 27
Tabulka 2 – Odlišnosti mezi jednotlivými tarify aplikace Postman 44
Tabulka 3 – Cenové porovnání jednotlivých tarifů 45
Tabulka 4 – Popis služeb 48
Tabulka 5 – Soubory WADL ke službám 48

SEZNAM ZKRATEK A ZNAČEK

API – Application Programming Interface

CD – Continuous Delivery

CI – Continuous Integration

CSV – Comma-Separated Values

E2E – End to End

GUI – Graphical User Interface

HTTP – Hypertext Transfer Protocol

HTTPS – Hypertext Transfer Protocol Secure

ICT – Information and Communications Technology

IDL – Interface Definition Language

JSON – JavaScript Object Notation

RAD – Rapid Application Development

REST - Representational State Transfer

RPC – Remote Procedure Call

RUD – Rational Unified Process

SDLC – Software Development Life Cycle

SOAP - Simple Object Access Protocol

TDD – Test-Driven Development

UAT – User Acceptance Testing

URI – Uniform Resource Identifier

URL – Uniform Resource Locator

XML – Extensible Markup Language

ÚVOD

V dnešní době se moderní aplikace skládají z mnoha různých komponent, které spolu musí navzájem spolupracovat a komunikovat, aby byla zajištěna bezproblémová funkčnost, a často se spoléhají na API. Integrační testování je nezbytnou součástí procesu testování softwaru. Zajišťuje správné fungování systému nebo softwarové aplikace testováním komunikace mezi jednotlivými komponentami, snižuje rizika odhalováním chyb v dřívějším stádiu vývoje softwaru a obecně zvyšuje kvalitu dodávaného produktu.

Teoretická část diplomové práce pojednává o současném stavu testování softwaru. Zabývá se historií testování, ve které rozebírá a charakterizuje jednotlivá období. Dále se věnuje statickým a dynamickým testovacím technikám. V následujícím úseku popisuje, jak probíhá testování v rámci životního cyklu vývoje softwaru dle nejpoužívanějších modelů. Závěrem první kapitoly se práce zabývá základními úrovněmi testování, a to testováním jednotek, integračním testování, systémovém testování a akceptačním testování.

Ve třetí kapitole je popsán HTTP protokol, který slouží jako komunikační protokol mezi klientem a serverem, jsou zde uvedeny metody HTTP požadavků a stavové kódy. Další část práce se věnuje aplikačnímu rozhraní umožňujícímu komunikaci mezi aplikacemi. Charakterizuje nejčastěji používané typy aplikačního rozhraní. Součástí třetí kapitoly je také charakteristika nástroje Postman, včetně jeho klíčových vlastností, funkcí a alternativ.

Diplomová práce si klade za cíl vytvořit manuál k nástroji Postman, který bude sloužit uživateli jako příručka pro využití této aplikace k testování API. V rámci praktické části bude testování s využitím nástroje Postman znázorněno na veřejně dostupné API.

Hlavním důvodem pro výběr tohoto tématu diplomové práce je vysoká popularita a široké využití nástroje Postman pro testování API. Postman na první pohled může působit jako jednoduchý nástroj, ale skýtá množství užitečných funkcí, které lze využít nejen pro automatizaci testování.

1 POPIS SOUČASNÉHO STAVU TESTOVÁNÍ SOFTWARU

V úvodní části této kapitoly je popsána historie testování a nejčastěji používané techniky testování softwaru. Dále se kapitola věnuje testování v rámci životního cyklu vývoje softwaru. V závěru kapitoly jsou popsány jednotlivé úrovně testování.

1.1 Historie testování softwaru

Testování softwaru je činnost, která se zrodila na počátku 2. poloviny 20. století, kdy došlo k rozvoji první generace počítačů a s nimi k nárůstu vývoje softwarových programů. Od tohoto období se testování neustále vyvíjelo a měnilo v čase v závislosti na rozvoji ICT technologií.

David Gelperin a Bill Hetzel rozdělují historii testování do pěti významných období. [1]

1.1.1 Období ladění

Období ladění (The Debugging-oriented period) probíhalo na začátku 50. let 20. století do roku 1956. V tomto období ještě nebylo rozlišováno mezi pojmy testování a ladění. Dané pojmy nebyly dostatečně vymezeny a bylo považováno za obtížné je přesně definovat. Programátoři vyvíjeli software, který si zároveň sami otestovali. V případě nalezených defektů zjistili jejich příčinu a defekt odstranili. Pojmy tester a testování byly v tomto období neznámé. [1]

1.1.2 Demonstrační období

V roce 1957 během demonstračního období (The Demonstration-oriented period) došlo k rozlišení pojmů ladění a testování Charlesem Bakerem v jeho recenzi na knihu Digital Computer Programming, jejímž autorem byl Danem McCrackenem.

Testování mělo dva hlavní cíle:

- zajistit, že software běží,
- zajistit, že software splňuje požadavky (vyřeší problém).

Testování začalo nabývat na významu vzhledem k nárůstu počtu aplikací, jejich rostoucí ceně a komplexitě. Manažeři začali věnovat oblasti testování více úsilí, aby bylo efektivnější a dokázalo lépe odhalilo defekty před nasazením aplikací do produkčního prostředí. [1]

1.1.3 Destruktivní období

V destruktivním období (The Destruction-oriented period) v letech 1979–1982 bylo testování považováno za destruktivní proces spouštění programu s cílem nalézt jeho chyby. Tuto definici zmínil jako první Glenford J. Myers ve své knize The Art of Software Testing. [1]

1.1.4 Období orientované na hodnocení

V roce 1983 publikoval Institute for Computer Sciences and Technology doporučený postup pro testování popisující metodologii, která integruje jednotlivé činnosti testování v průběhu životního cyklu vývoje softwaru, aby bylo možné hodnotit a měřit kvalitu vytvořeného softwaru. V období orientovaném na hodnocení (The Evaluation-oriented period) byl produkt testován do té doby, dokud nebylo dosaženo přijatelného ohodnocení kvality softwaru (tj. počet nalezených defektů a jejich závažnost byla v přijatelné míře). [1]

1.1.5 Období zaměřené na prevenci

Během období zaměřeném na prevenci (The Prevention-oriented period) v letech 1988–2000 vyšel do popředí nový přístup, při kterém se testovací aktivity zaměřovaly na 3 oblasti:

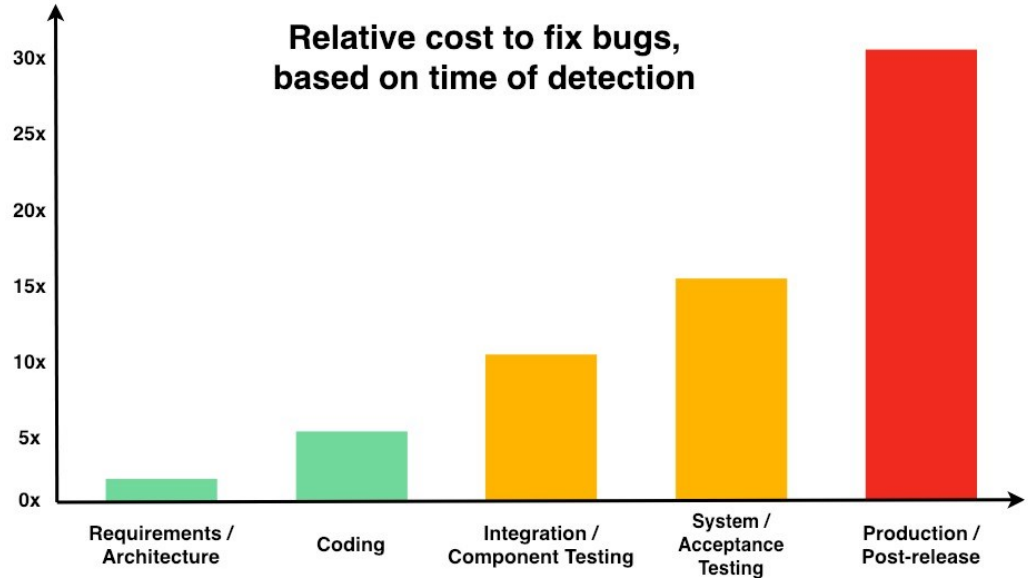
- prokázat, že software splňuje softwarovou specifikaci,
- nalézt defekty,
- předejít defektům.

Preventivní model zařazuje testovací aktivity jako součást risk managmentu, jelikož v případě správného přístupu dokáže identifikovat a významně snížit projektová rizika. Vychází z principu, že čím dříve dojde k nalezení defektu, tím snadnější a levnější bude jeho oprava. Testování je plánováno ve dvou fázích. Hlavní testovací plán je sestaven na začátku projektu, ohodnocuje rizika a definuje testovací strategii, zatímco detailní plán je oddělená aktivita reagující na schválené požadavky. [2]

1.2 Statické testovací techniky

Statické testování je testovací technika, která nevyžaduje spouštění softwaru. S pomocí statických technik je možné začít s testováním softwaru již v raných fázích životního cyklu a odhalit případné defekty. V případě, že by tyto defekty byly nalezeny až v dalších fázích vývoje, jejich oprava by byla časově náročnější a výrazně nákladnější. Je zde aplikováno pravidlo, že čím dříve je možné identifikovat chybu, tím levnější by měla být její oprava.

Obrázek 1 znázorňuje porovnání relativních nákladů na opravu chyby v závislosti na tom, v jaké fázi byla chyba objevena. Z obrázku je zřejmé, že nejnákladnější je oprava chyby, která byla detekována na produkčním prostředí. [3]



Obrázek 1 – Relativní náklady na opravu chyby dle času odhalení

Zdroj: [4]

1.2.1 Revize

Revize je manuální proces kontroly vytvořených pracovních produktů, jsou to mimo jiné:

- byznysové specifikace,
- uživatelské scénáře,
- návrhy architektury systému,
- návrhy GUI,
- testovací případy,
- testovací plány,
- testovací podmínky,
- testovací skripty apod.

Dle požadovaného cíle revize je nutné zvolit vhodný typ revize. Existuje několik základních typů revizí. [5]

a) Neformální revize

Neformální revize je hojně využívána při agilním vývoji na začátku životního cyklu produktu. Je prováděna zpravidla s blízkými kolegy z týmu a její výsledek běžně není zaznamenáván. Cílem neformální revize je odhalit možné defekty softwaru a konzultovat různá řešení nebo nápady. [5]

b) Předvedení (Walkthroughs)

Při předvedení je na revizní schůzce vytvořený pracovní produkt prezentován jeho autorem. Hlavním účelem tohoto typu revize je odhalení defektů, zlepšení produktu a zhodnocení míry shody s normami a specifikacemi. V rámci schůzky by měl být přítomen zapisovatel. [5]

c) Technická revize

Cílem technické revize je dosáhnout shody v pracovním produktu, zjistit potenciální defekty a budovat důvěru ve kvalitu vytvořeného produktu. Předpokládá se, že účastníci revize budou osoby, které jsou odborníky v dané nebo příbuzné oblasti či oboru. Zapisovatel zaznamenává seznam potenciálních defektů a tvoří report o průběhu revize. [5]

d) Inspekce

Inspekce je nejformálnějším typem revize. Důraz je kladen na dodržování předem daného procesu revizní schůzky s formálně zdokumentovanými výstupy dle určených pravidel (metriky, reporty, záznamy o defektech). Na schůzce má každý účastník vymezenou roli (autor, manažer, prostředník, vedoucí revize, revidující, zapisovatel), kterou dodržuje po celou dobu trvání revize. Cíle revizní schůzky jsou velmi podobné technické revizi. [5]

1.2.2 Statická analýza

Statická analýza je testovací technika, která zkoumá kód vytvořený vývojářem bez jeho spuštění. Statická analýza může být prováděna manuálně nebo s pomocí automatizovaného nástroje. Při statické analýze nejsou využívána žádná vstupní data. S pomocí statické analýzy lze například detekovat potenciální narušení bezpečnosti (SQL injection – narušení databáze) nebo logické nekonzistence kódu. [6]

Mezi největší benefity statické analýzy patří hloubka analýzy. Statická analýza dokáže pokrýt všechny cesty kódu, což v rámci dalších testovacích technik není vždy možné nebo z hlediska nákladů ekonomicky efektivní. V případě využití automatizovaného nástroje je výhodou rychlost a přesnost provedené analýzy.

Spuštěním nástroje lze provést revizi kódu takřka okamžitě. V návaznosti na výsledek revize a vyhodnocení chyb může vývojář nalezené defekty opravit. Vzhledem k tomu, že člověk je bytost omylná, automatizované nástroje dosahují vysoké přesnosti při odhalování chyb v kódu a jejich lokace, čímž zajišťují kvalitu v souladu s danými standardy programovacího jazyka. Následné dynamické testování bude díky vyšší kvalitě kódu efektivnější a bude vyžadovat méně úsilí. [7]

Mezi používané nástroje pro statickou analýzu kódu patří například Raxis, PVS-Studio, SonarQube, DeepSource nebo Embold. [8]

1.3 Dynamické testovací techniky

Dynamické testování se od statického liší v tom, že při něm dochází ke spouštění kódu softwaru. Mezi dynamické testovací techniky se řadí black-box testování, white-box testování a testování založené na zkušenostech.

1.3.1 Black-box testování

Black-box testování (testování metodou černé skříňky) se zaměřuje na aspekty aplikace z hlediska její funkčnosti dle požadované specifikace produktu.

Testovací techniky:

- analýza hraničních hodnot (BVA – boundary value analysis),
- rozhodovací tabulky (decision tables),
- přechody stavů (state transition),
- případy užití (use cases). [9]

Tester na danou aplikaci nahlíží jako na černou skříňku, kdy nezná její obsah a pouze s ní pracuje, provádí testy a vyhodnocuje výsledky. Na obrázku č. 2 je zobrazen ilustrační obrázek metody testování černé skříňky. Za černou skříňku je považován software.

Obrázek 2 – Black-box testování

Zdroj: vlastní zpracování v programu draw.io

1.3.2 White-box testování

White-box testování (testování metodou bílé skříňky) zkoumá vnitřní strukturu a logiku programu. K testování je obvykle vyžadován kvalifikovanější tester vzhledem k nutnosti hlouběji porozumět vnitřní struktuře programu (programovacímu jazyku).

Užívané testovací techniky:

- pokrytí příkazů,
- pokrytí rozhodnutí,
- vícenásobné testování podmínek (multiple condition testing),
- API testování.

Jednotlivé white-box techniky umožňují systematické odvození testovacích případů a zaměřují se na konkrétní aspekty struktury programu. [9]

1.3.3 Testování založené na zkušenostech testera

Pokud se tester rozhodne využít techniku testování založenou na zkušenostech testera, může použít například techniku průzkumného testování nebo odhalování chyb.

Průzkumné testování (Exploratory testing) je testovací technika běžně používaná v agilním vývoji. V průběhu testování testeři zkoumají aplikaci, aby identifikovali a zdokumentovali potenciální chyby. Tester musí porozumět požadavkům na software, který testuje. Testerovi je poskytnuta svoboda provést testování dle vlastního uvážení, přičemž není omezen testovací sadou a předem danými testovacími scénáři. [10]

Odhadování chyb (Error guessing) je testovací technika, kterou tester používá takovým způsobem, že vymýšlí situace, v nichž se software nemusí chovat správně a může obsahovat defekty vedoucí k jeho selhání. Klíčem k testování je odhalit kritická místa v softwaru. Technika odhadování chyb vyžaduje zkušeného testera, který má zkušenosti v dané oblasti a zná časté typy chyb vývojářů. Techniku odhadování chyb je vhodné použít v situacích, kdy pro fázi testování z různých důvodů není alokováno dostatečné množství kapacit nebo nezbývá adekvátní množství času. [11]

1.4 Testování v rámci životního cyklu vývoje softwaru

Životní cyklus vývoje softwaru (SDLC) je proces používaný k navrhování, vývoji a testování softwaru. SDLC si klade za cíl vytvořit vysoce kvalitní software, který zcela naplňuje očekávání

zákazníků, je dodán ve smluveném čase a jeho cena odpovídá předpokládaným nákladům. Životní cyklus projektu je jeho průběh, který se dělí do několika částí dle společných charakteristik. Modelování definuje počátek a životní cykly objektů či procesů a dalších komponent, což napomáhá ke správnému navržení softwaru. Modelování je považováno za podpůrný prostředek tvorby dokumentace.

Vývoj softwaru je zpravidla týmová práce, při které je nutné zkoordinovat a zorganizovat všechny činnosti a delegovat je na správné pracovníky týmu. V softwarovém týmu jsou zpravidla obsazeny pracovní pozice manažera, analytika, architekta, vývojáře, testera, správce IT infrastruktury apod. [12]

V následujících podkapitolách budou představeny vybrané modely životního cyklu vývoje softwaru se zaměřením na testovací fáze.

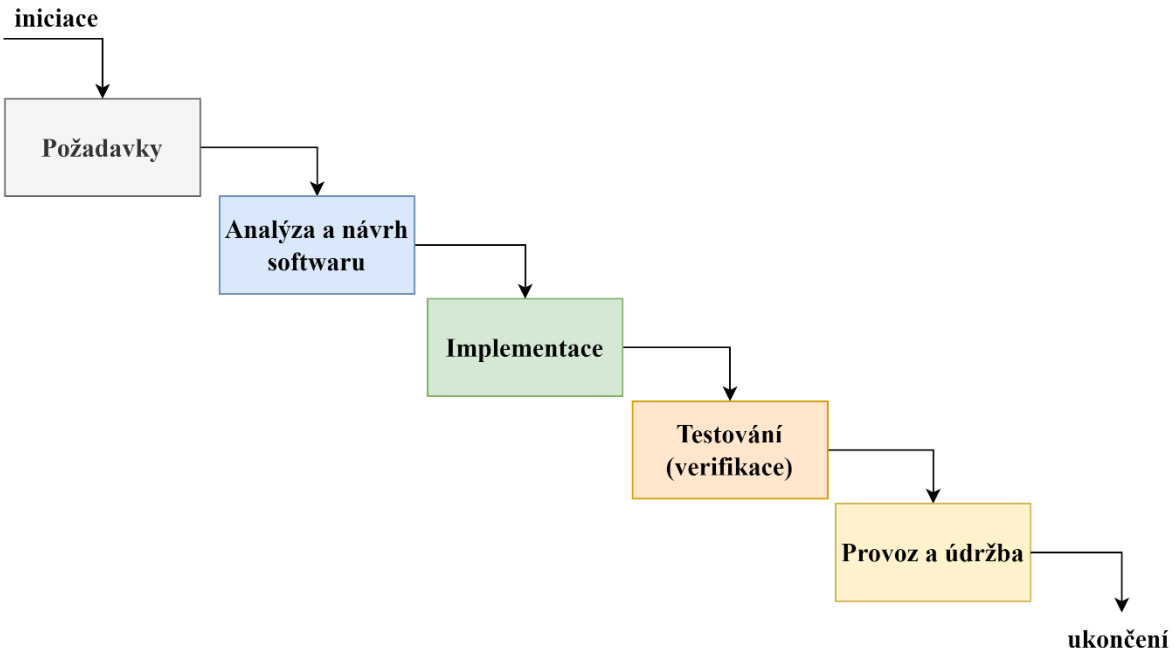
1.4.1 Vodopádový model

Vodopádový model (Waterfall model) je systematický model založený na strukturované metodice, vhodný pro přesně definované systémy se stálými požadavky. Projekt musí být detailně a precizně naplánován.

V průběhu životního cyklu na sebe jednotlivé etapy plynule navazují. Jeho nevýhodou je dlouhá doba trvání vývoje, jelikož první verze produktu jsou k dispozici až v závěrečných etapách, v nichž zároveň dochází k prvnímu testování produktu. [12]

„Testování se provádí pouze na konci vývojového cyklu, a proto může ze začátku vývoje dojít k podstatnému problému, na který se přijde až v okamžiku, kdy se podle rozvrhu prací má již hotový produkt umístit na trh.“ [13]

Z tohoto důvodu je i oprava nalezených defektů při fázi testování složitější a nákladnější. Model nepočítá s cyklickými návraty do přechozích etap. Význam modelu je především teoretický.



Obrázek 3 – Schéma vodopádového modelu

Zdroj: vlastní zpracování dle [12]

1.4.2 Spirálový model

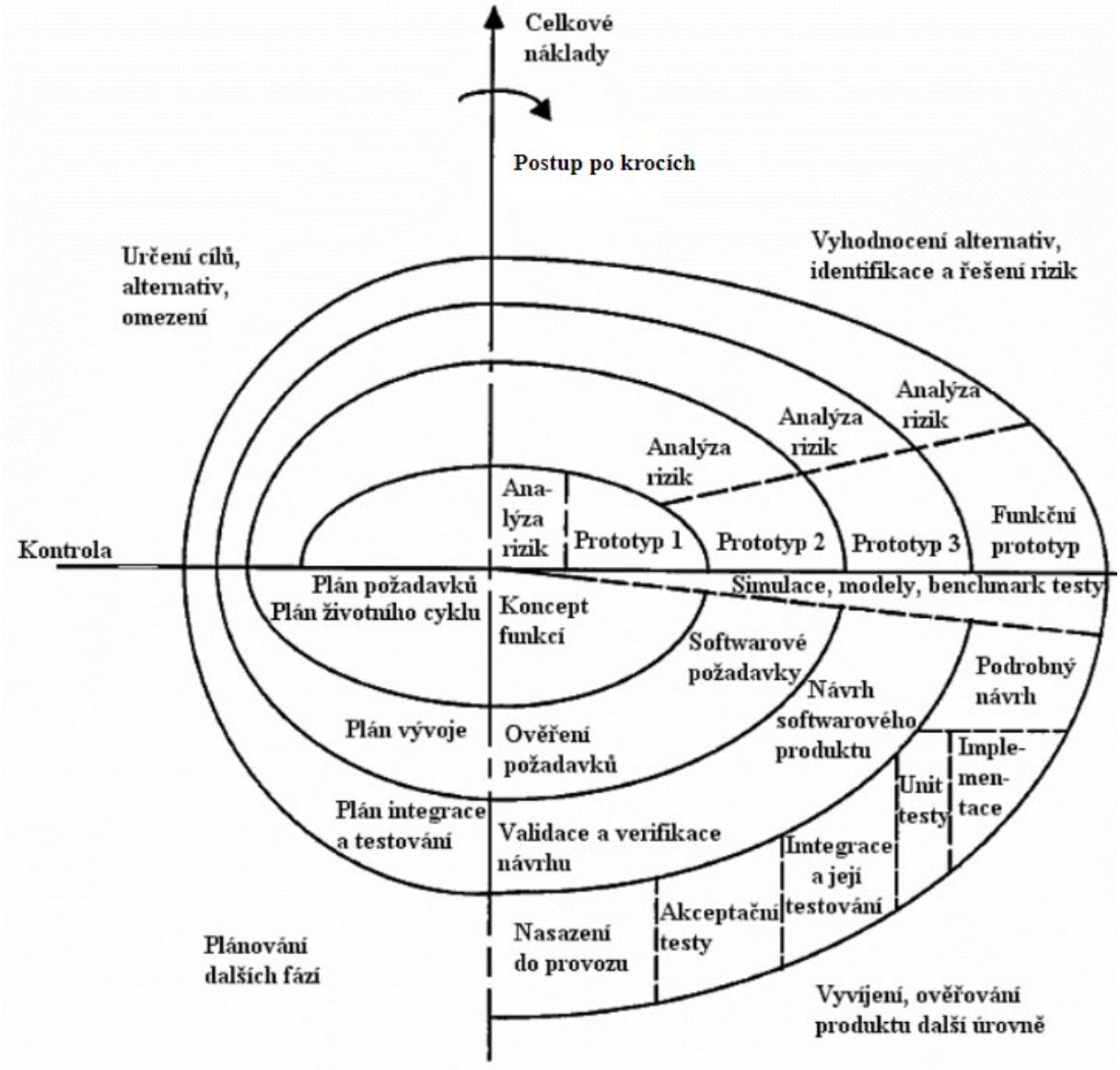
Spirálový model popsal Barry Boehm ve svém článku „A Spiral Model of Software Development and Enhancement“ v roce 1986. Cílem článku bylo odstranit nedostatky vodopádového životního cyklu, přičemž se Barry Boehm zaměřil především na problematiku průběžných změn požadavků.

Spirálový model je iterativní model, který vychází z principů již zmíněného vodopádového modelu a prototypového modelu. Nadto je na začátku každé etapy uskutečněna důkladná analýza rizik.

Spirálový model je možné rozdělit do 4 částí. První část obsahuje stanovení omezení, cílů a alternativ, které jsou následně posouzeny. Druhá část se zabývá analýzou rizik. Třetí část obsahuje vývoj a verifikaci další úrovně produktu. Ve čtvrté části se plánují další etapy vývoje produktu, pokud v přechozí etapě nedošlo k jeho dokončení. [14]

Software je testován pravidelně. Součástí každé etapy je testování softwaru a reportování současného stavu, díky čemuž jsou defekty nalezeny včas. V případě velkého množství defektů může být analýza rizik upravena. [15]

Schéma spirálového modelu je zobrazeno na obrázku 4.



Obrázek 4 – Spirálový model

Zdroj: [15]

Nevýhodou spirálového modelu je jeho vysoká nákladnost, s větším rádiem spirály náklady stoupají, proto je vhodnější spíše pro velké projekty. Model je komplexnější ve srovnání s ostatními modely a k jeho aplikaci je potřeba zkušený tým.

Z hlediska projektového managementu je složité odhadovat časovou náročnost jednotlivých činností či etap. Projekt je závislý na provádění kvalitní analýzy rizik.

Výhodou spirálového modelu je možnost získat zpětnou vazbu od zákazníků v průběhu projektu a určitá míra flexibility reagovat na změnové požadavky po zahájení vývoje. [16]

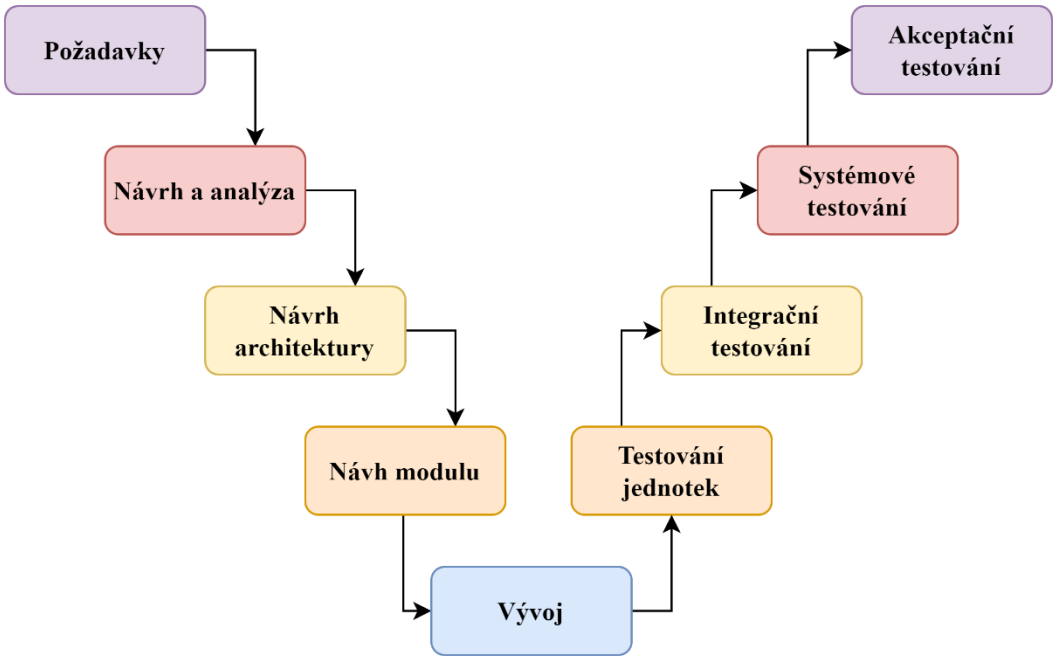
1.4.3 V-model a W-model

V-model, také známý jako verifikační a validační model, je model, ve kterém jsou procesy prováděny sekvenčně jeden po druhém ve tvaru písmena V. Jedná se o rozšířený vodopádový model, který je v praxi poměrně používaný.

U V-modelu se oproti vodopádovému modelu lze vracet do předchozích etap, je flexibilnější a nákladnější. Testování je započato již v první fázi, je tedy možné testovat v průběhu vývoje produktu. Běžně je nalezeno vyšší množství defektů, čímž se zároveň zvyšuje šance na úspěšnost projektu. [17]

Jednotlivým vývojovým fázím vždy odpovídá konkrétní testovací fáze. Například specifikaci požadavku odpovídá akceptační testování. [18]

Schéma V-modelu je zobrazené na obrázku č. 5.

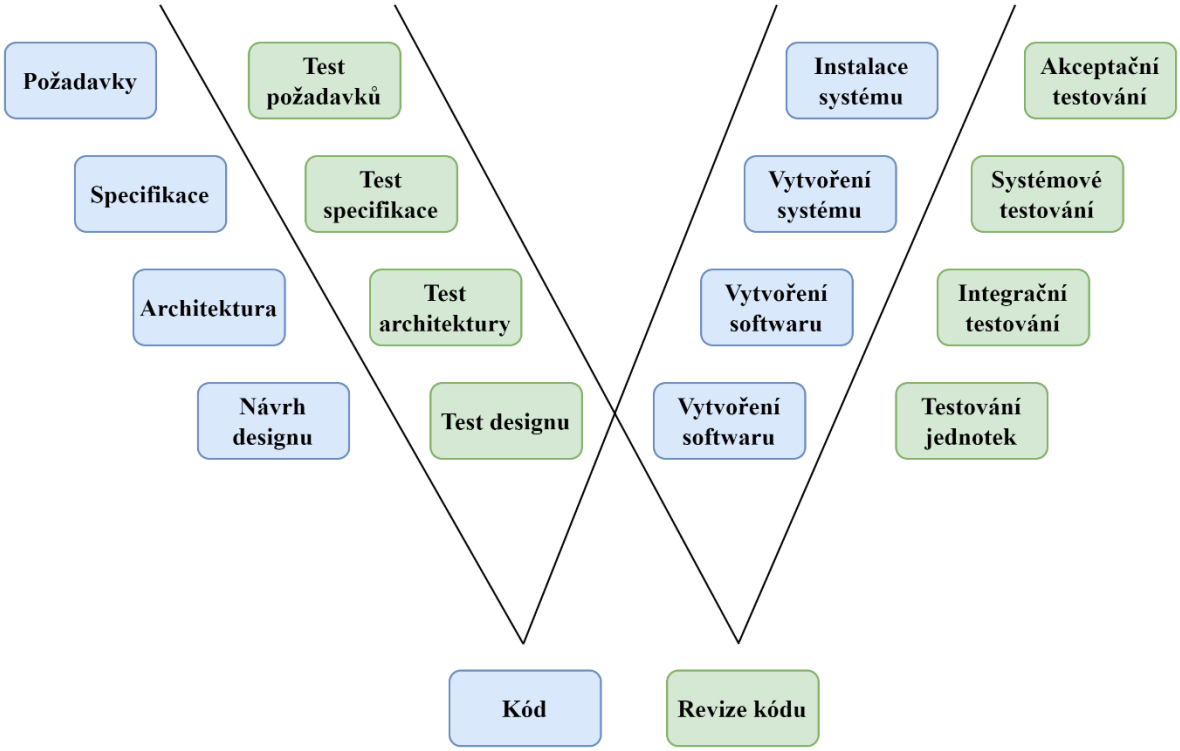


Obrázek 5 – V-model

Zdroj: vlastní zpracování v programu draw.io

Alternativou k V-modelu je W-model, který je jeho rozšířenou verzí, přičemž je kladen větší důraz na testovací aktivity, které jsou vykonávány paralelně s vývojovými aktivitami. [19]

Na obrázku č. 6 je zobrazeno schéma W-modelu.



Obrázek 6 – W-model

Zdroj: vlastní zpracování v programu draw.io dle [19]

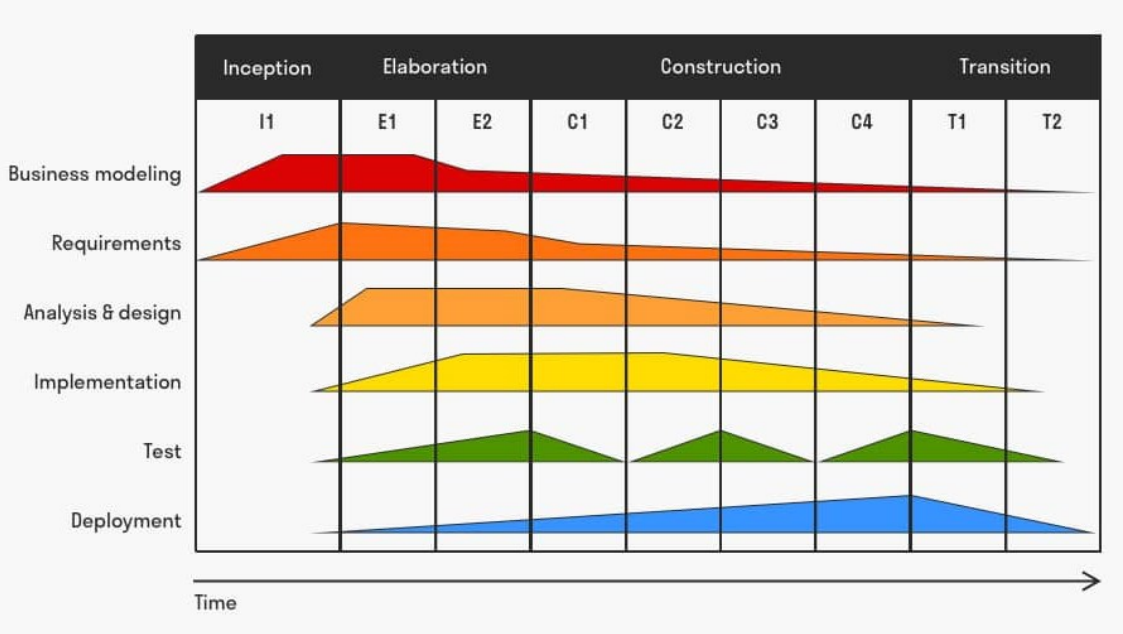
1.4.4 Rational Unified Process model

Rational Unified Process (RUP) model je rozšířenou verzí UP (Unified Process) modelu, který se liší v jednotlivých detailech metodik, nikoliv v sémantice. Jedná se o agilní model vývoje softwaru, který je rozdělen do 4 etap:

- počátek (inception),
- rozpracování (elaboration),
- realizace (construction),
- zavedení (transtition).

Během těchto etap probíhají různé činnosti (business modelování, zpracování systémových požadavků, analýza a návrh, implementace, testování a nasazení podporované projektovým managementem, správou změnových požadavků a používaných IT prostředí).

Dané činnosti se v průběhu opakují, dokud produkt nesplňuje zadané požadavky, tj. jde o iterativní model. Na následujícím obrázku č. 7 je zobrazen průběh modelu na vodorovné ose z hlediska času a na svislé ose z hlediska vykonávaných činností. [20]



Obrázek 7 – Rational Unified Process model

Zdroj: [20]

Jednotlivé iterace mohou trvat až několik měsíců, z tohoto důvodu je objem nově dodaných vlastností softwaru velký, z čehož vyplývá vyšší náročnost následného testování. Agilní modely obecně nejsou vhodné pro komplexní a robustní systémy, využití najdou především v proměnlivých prostředích. [21]

1.4.5 Scrum model

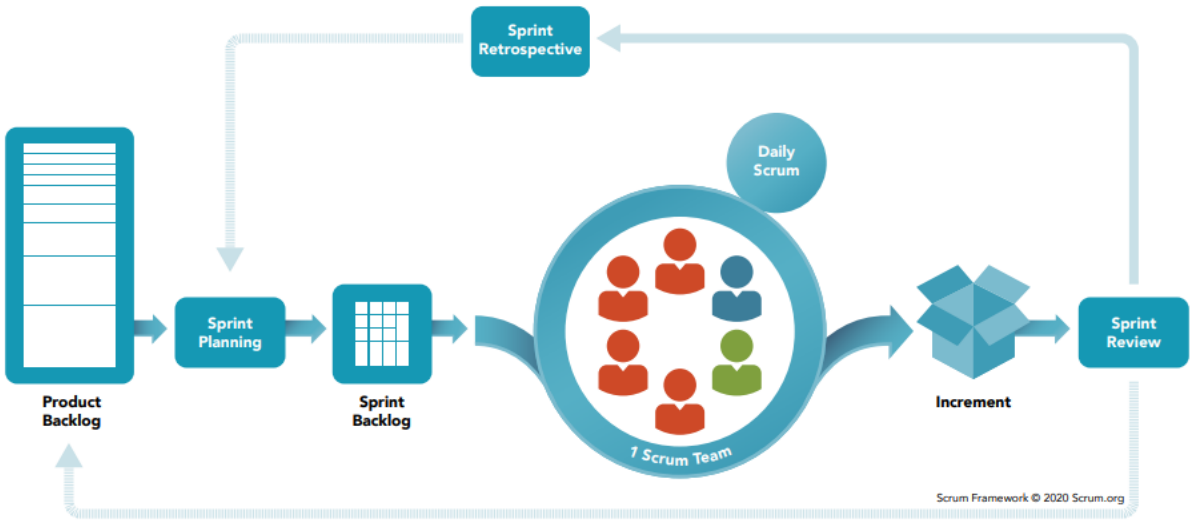
Scrum model lze definovat jako iterativní, inkrementální a agilní model vývoje softwaru. Scrum lze také popsat jako rámec projektového řízení, který je vhodný pro projekty s komplikovanými nebo jedinečnými požadavky.

Scrum je složen ze série iterací (sprintů). Délka jednoho sprintu není pevně stanovená, obvykle sprint probíhá 2 až 4 týdny. Účastníci scrumu pracují v týmu, který má jasně stanovené role. Jedná se například o roli vlastníka produktu (product owner), scrum mastera nebo role vývojového týmu. [22]

Na začátku každé iterace je nezbytné daný sprint naplánovat, tj. určit, které položky z product backlogu budou realizovány a vloženy do sprint backlogu. Na začátku každého pracovního dne ve stejnou dobu se koná patnáctiminutový denní scrum, během něhož je sprint revidován a upravován.

Po sprintu je zhodnocen jeho výsledek, který je následně prezentován zainteresovaným stranám. V průběhu retrospektivy sprintu se hodnotí poslední sprint (stanovené procesy, použité nástroje a technologie) a rozhoduje se o dalším postupu prací v navazujících sprintech. [22]

Kompletní proces scrum modelu je zobrazen na obrázku č. 8.



Obrázek 8 – Scrum model

Zdroj: [23]

Testeři se účastní plánování sprintu, denních scrumů a spolupracují na retrospektivě sprintu. Testování je prováděno v průběhu celého sprintu a je reportováno stakeholderům s pomocí různých testovacích metrik (např. sprint goal success, escaped defects, defect density, team velocity). Vzhledem k hojným menším přírůstkům softwaru v průběhu projektu využívá velké míry automatizace v různých úrovních testování. [24]

1.4.6 Rapid application development model

Rapid Application Development (RAD) model je lineární sekvenční model vývoje softwaru založený na prototypování. Klade důraz na včasné testování prototypu zákazníkem.

Jednotlivé části softwaru (moduly) jsou vyvíjeny paralelním způsobem ve formě menších projektů. Postupně dochází k jejich skládání do funkčního softwarového prototypu.

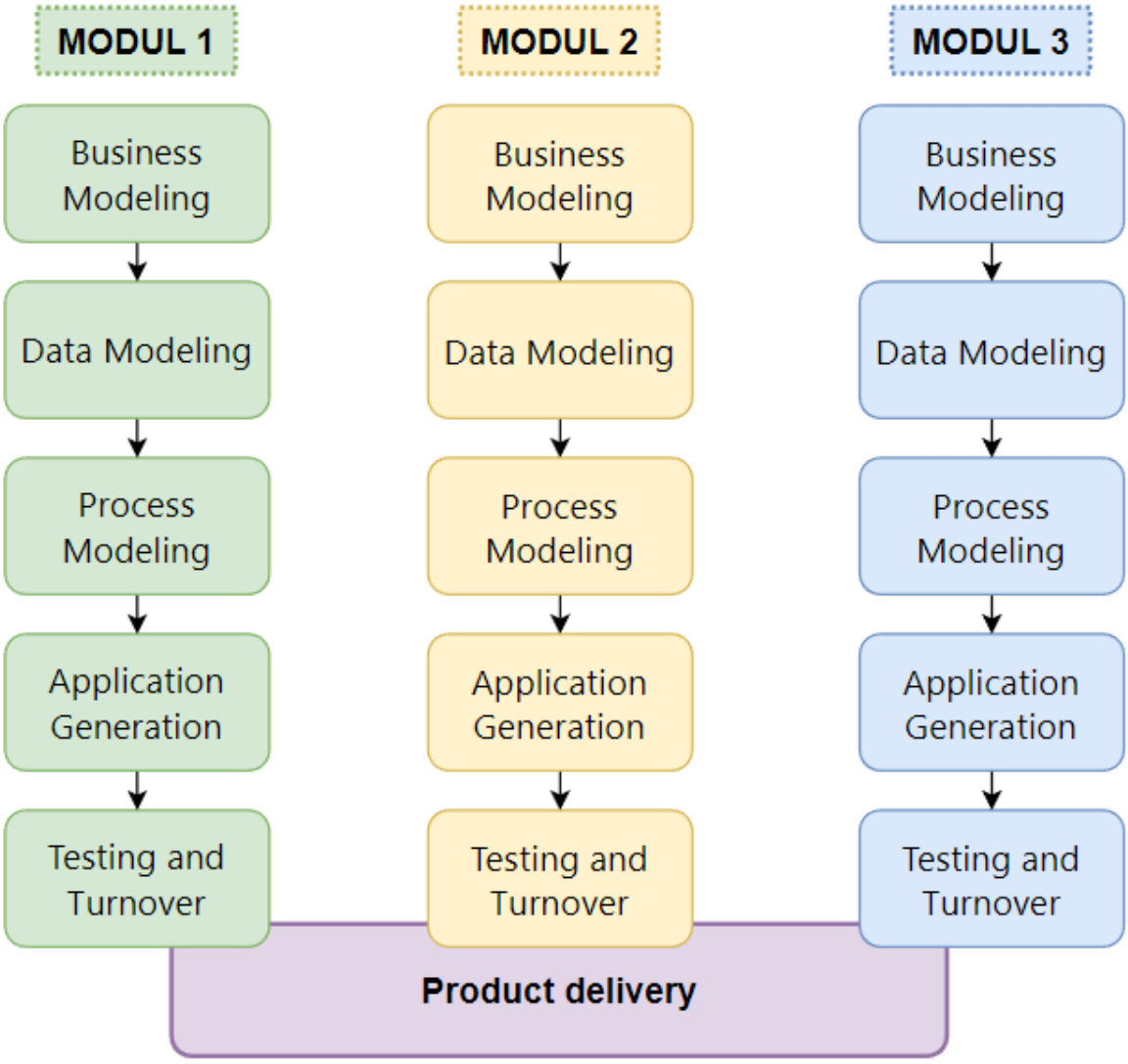
Zcela zásadní je účast zástupců byznysu v průběhu celého životního cyklu projektu, a to zejména při testování vytvořených prototypů a při korekci požadavků. [25]

RAD model je složen z pěti hlavních etap:

- byznys modelování (business modeling),

- modelování dat (data modeling),
- procesní modelování (process modeling),
- generování aplikací (application generation),
- testování (testing and turnover). [26]

Na obrázku č. 9 je zobrazen životní cyklus RAD modelu.



Obrázek 9 – Rapid application development model

Zdroj: vlastní zpracování v programu draw.io dle: [26]

Předpokladem pro úspěšně použití tohoto modelu je znalost požadavků, dostatečný rozpočet a možnost vytvořit software, který lze modularizovat. Výhodou aplikace tohoto modelu je rychlé vytvoření koncového produktu, k čemuž přispívá možnost opakovaně využívat již

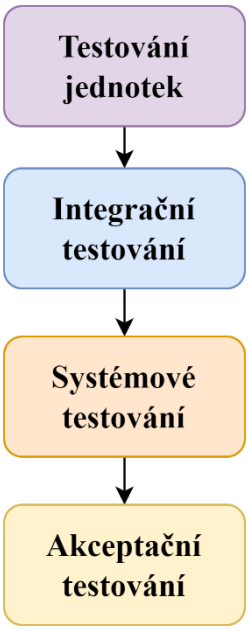
vytvořené komponenty, což zkracuje celkový čas potřebný k vývoji produktu. Díky kratším iteracím je snazší vyhovět měnícím se požadavkům v průběhu vývoje.

Příliš vysoká složitost modularizace ovšem může vést k selhání celého projektu. Použití nástrojů, které značně usnadňují vývojářům práci, vyžaduje vysoce kvalifikované zaměstnance. RAD model není vhodný pro menší projekty, a to zejména v případě, kdy cena použitých nástrojů je neúměrná k rozpočtu projektu. [27]

1.5 Úrovně testování

Aby bylo možné dosáhnout úspěšného a efektivního otestování softwaru během SDLC, existuje pro každou úroveň vývoje adekvátní úroveň testování. Každá úroveň testování má svůj konkrétní účel, příznačné defekty, specifické postupy a poskytuje přidanou hodnotu k testování v průběhu SDLC. V následujících podkapitolách budou popsány úrovně testování užívané v oblasti systémového inženýrství, jmenovitě testování jednotek, integrační testování, systémové testování a akceptační testování. [28]

Posloupnost úrovní testování je zobrazena na obrázku č. 10.



Obrázek 10 – Úrovně testování

Zdroj: vlastní zpracování v programu draw.io

V tabulce č. 1 je zobrazeno porovnání jednotlivých úrovní z hlediska toho, jací pracovníci jsou za danou úroveň testování zpravidla zodpovědní, a to z hlediska cílů a zaměření dle oblastí.

Tabulka 1 – Porovnání úrovní testování

Úroveň testování	Vlastník	Cíle	Klíčové oblasti
Testování jednotek	Vývojářský tým	Detekce chybného, nezabezpečeného a neudržitelného kódu. Snížení rizika selhání jednotky.	Funkčnost, bezpečnost a udržovatelnost
Integrační testování	Testovací tým (zřídka vývojový tým)	Detekce defektů v rozhraní jednotek. Snížení rizika selhání toku dat.	Funkčnost, kvalita dat, interoperabilita a kompatibilita jednotek, výkon
Systémové testování	Testovací tým	Detekce defektů v případech užití a end-to-end scénářích. Snížení rizika nesplnění byznys požadavků.	Funkčnost, kvalita dat, bezpečnost, výkon, spolehlivost, použitelnost, udržovatelnost, instalovatelnost, přenositelnost, interoperabilita
Akceptační testování	Uživatelský tým	Ověřit připravenost softwaru k nasazení. Ověřit scénáře z uživatelského pohledu	Funkčnost, použitelnost

Zdroj: vlastní zpracování v programu Microsoft Excel dle [28]

1.5.1 Testování jednotek

Testování jednotek (unit testing) nebo také testování modulů je testování na nejnižší úrovni. Jednotkou se rozumí izolovaná, nejmenší testovatelná část softwaru, kterou je možné spustit. Cílem jednotlivých testů je ověřit, zda každá komponenta správně plní svou funkci

dle specifikace. Typickými chybami může být nesprávná logika kódu nebo nesoulad funkčnosti jednotky v souvislosti se zadáním. [13]

V případě, že je nalezen defekt během testování, je velmi pravděpodobné, že se chyba nachází v dané jednotce. Tato skutečnost vede k mnohonásobně účinnějšímu debuggingu (ladění) a jednoduššímu konfirmačnímu testování. Protože byl defekt nalezen v rané fázi vývoje, postačí provést retestování pouze u dané jednotky, u které byl defekt nalezen.

Oproti dalším úrovním u testování jednotek obvykle nedochází k běžnému reportování chyb, jelikož nalezené chyby jsou programátorem opraveny obratem a jejich dokumentace není nezbytná. Testování jednotek provádí převážně přímo vývojáři a často si pro tento účel vytváří automatizované testy. Pouze v ojedinělých případech testy vykonávají testeři, v takovém případě potřebují přístup ke kódu aplikace. [13]

Přestože se testování jednotky běžně provádí až po jejím naprogramování, v případě vývoje řízeném testováním (TDD) jsou automatizované testy sestaveny ještě před vývojem jednotky. První aktivitou vývojáře je v takovém případě napsat test jednotky, který po spuštění selže. [29]

Autoři Matt Stephens a Doug Rosenberg ve své knize Testování softwaru řízené návrhem zmiňují, že užití metodiky TDD v praxi není jednoduché.

„Čistý efekt postupování podle metodiky TDD spočívá v tom, že vše dostane vlastní test jednotky. Teoreticky to sice zní skvěle, ale prakticky budete mít hrozně moc redundantních testů. Metodika TDD má image „odlehčeného“ nebo též agilního procesu, protože se vyhýbá představě ‚velkého návrhu na začátku‘, což může nalákat k dříve zahájenému programování. Jenže tato iluze rychlého úspěchu je brzy odsunuta těžkou prací ve formě refaktorování až k hotovému produktu, v jejímž průběhu se přepisuje kód i testy.“ [29]

Benefity pro projekt při testování jednotek:

- zvýšení důvěry v produkt při častých změnách, v případě dobře napsaných automatizovaných regresních testů,
- znovupoužití jednotek,
- rychlejší vývoj, jelikož čas strávený psaním jednotkových testů je nižší než čas strávený laděním defektů nalezených v dalších úrovních testování,
- velmi nízké náklady na opravu chyb,
- spolehlivost kódu. [30]

1.5.2 Integrační testování

Integrační testování je úroveň testování vývoje softwaru, která detekuje defekty ve vztazích mezi jednotkami nebo jednotlivými systémy. Dle této definice se integrační testování dále dělí na integrační testování jednotek a systémové integrační testování. Běžný softwarový produkt se skládá z mnoha jednotek, v rámci jejich integrace jsou tyto jednotky sestavovány dohromady do subsystémů, dále do systémů. Je možné, že jednotky fungovaly perfektně izolovaně, ale při jejich integraci došlo k selhání. Z tohoto důvodu je během integračního procesu vhodné provádět integrační testování. [31]

Důvodů, proč provádět integrační testování, je celá řada, mimo jiné:

- nekompatibilita mezi jednotlivými moduly může způsobit selhání softwaru,
- mohou nastat potenciální problémy s kompatibilitou hardwaru,
- nutnost ověření interakce s API nebo systémy třetích stran. [31]

Benefity integračního testování:

- integrační testování zajišťuje správnou funkci integrovaných jednotek nebo systémů,
- integrační testování odhaluje defekty v rozhraní, databázích, infrastruktuře,
- detekce bezpečnostních incidentů,
- komplexní ověření konektivity celého systému, což výrazně snižuje pravděpodobnost vážných defektů. [31]

Předpokladem pro zahájení integračního testování je otestování jednotek v předchozí úrovni testování, přičemž všechny závažné defekty jsou vyřešeny, proběhla integrace jednotek, modulů nebo systémů a testovací tým má vytvořený testovací plán, testovací případy a připravené testovací prostředí.

Výstupními kritérii pro tuto testovací úroveň je úspěšné otestování integrovaného systému, zdokumentování výsledků testování, reporting výsledku provedené exekuce testovacích případů a opravení nalezených závažných defektů (dle předem stanovených kritérií přijatelnosti). [32]

Testování Application Programming Interfaces

API testování je typem integračního testování, kdy je testováno přímo API daného systému. API lze testovat manuálně, kdy tester používá testovací nástroj a manuálně ověřuje funkčnost endpointů API nebo automatizovaně s pomocí vytvořených testovacích skriptů. [33]

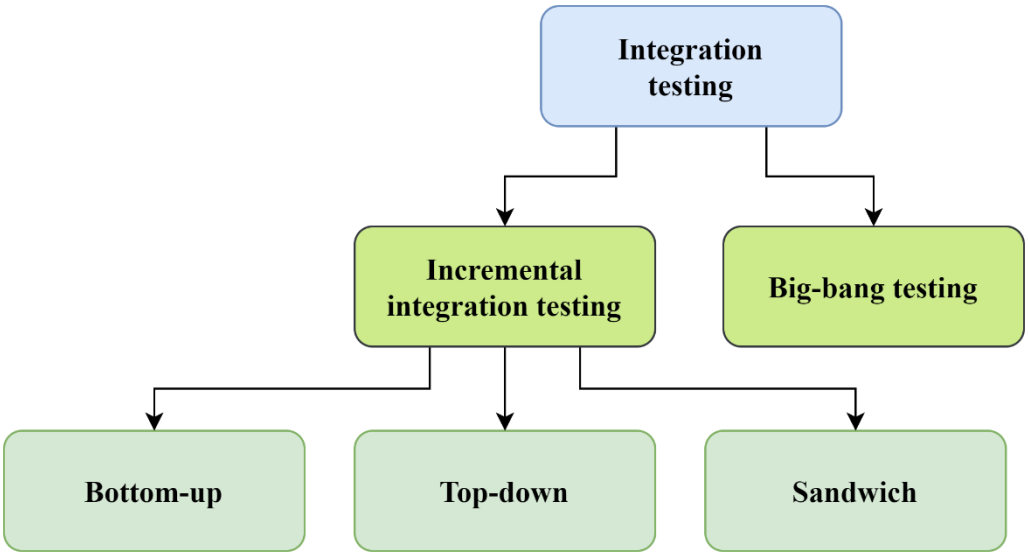
Testovat API lze různými způsoby:

- Funkční testování ověřuje, že endpoint vrací očekávanou odpověď dle odeslaného požadavku.
- Regresní testování je využíváno pro kontrolu, aby nově vyvinuté změny nezpůsobily defekty v testované aplikaci.
- Zátěžové testování ověřuje, zda je API schopná zpracovat velké množství požadavků.
- Bezpečnostní testování si klade za cíl ověření bezpečnostní zranitelnosti API zejména v oblastech neoprávněného přístupu a ochrany dat.
- Fuzzing metoda automaticky generuje a odesílá náhodná data, čímž se snaží odhalit možné zranitelnosti API. Metoda je vhodná pro API, která přijímají různé vstupy od uživatelů. [33]

Miroslav Bureš ve své knize Efektivní testování softwaru uvádí, že existují dva způsoby pro provádění automatických testů:

„1. Integrační testy napsané pomocí frameworku pro jednotkové testy. V testovaném systému voláme v sekvenci metody jednotlivých API a v jednotkovém testu sledujeme jejich interakci a funkčnost. Zajímá nás jak výsledek celého procesu, tak kontrola jednotlivých mezi výstupů. 2. Samostatný nástroj pro integrační testování, např. SoapUI nebo řada dalších.“ [34]

Další možné přístupy k integračnímu testování děleny do dvou kategorií jsou zobrazeny na obrázku 11. Jedním z přístupů je inkrementální integrační testování (incremental integration testing), které se dále dělí na přístupy bottom-up, top-down a sandwich. Druhou kategorií je big-bang integrační testování. [35]

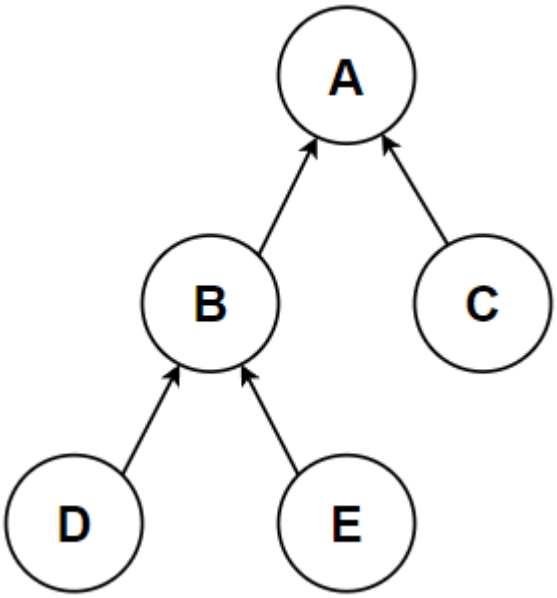


Obrázek 11 – Přístupy k integračnímu testování

Zdroj: vlastní zpracování v programu draw.io

Inkrementální integrační testování

Bottom-up, znázorněný na obrázku 12, je přístup k integračnímu testování, při kterém jsou nejprve testovány jednotky z hlediska architektury nejnižší úrovně ve směru k úrovním vyšším neboli zdola nahoru. Tento přístup je považován za nejefektivnější. [35]

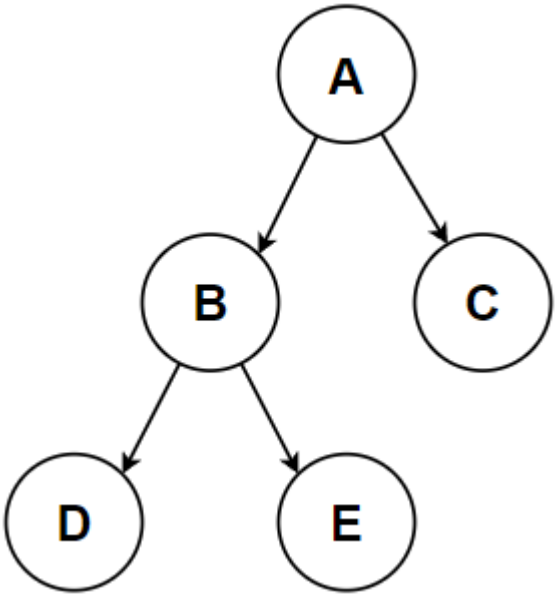


Obrázek 12 – Bottom-up přístup

Zdroj: vlastní zpracování v programu draw.io

Dle obrázku č. 12 by při použití této metody bylo testováno jako první rozhraní mezi jednotkami B a D a B a E, následovaly by testy rozhraní A a B a A a C.

Top-down přístup, znázorněný na obrázku 13, je protichůdným přístupem, při jehož užití jsou nejdříve testovány z hlediska architektury jednotky nejvyšší úrovně ve směru k úrovním nižším neboli shora dolů. [35]



Obrázek 13 – Top-down přístup

Zdroj: vlastní zpracování v programu draw.io

Dle obrázku č. 13 by při použití této metody bylo testováno nejdříve rozhraní mezi jednotkami A a B a A a C, následovaly by testy rozhraní B a D a B a E.

Sandwich, který kombinuje přístupy Top-down a Bottom-up, je třetím přístupem inkrementálního integračního testování. Tato metoda začíná zároveň na nejnižší i nejvyšší úrovni a jejím cílem je dosáhnout prostřední jednotky z obou směrů. [35]

Big-bang testování

Big-bang testování se běžně provádí na rozdíl od inkrementálního testování až v situaci, ve které jsou všechny moduly kompletně vyvinuty. Teprve po integraci všech součástí systému se provádí testování. Tento přístup se hodí spíše pro drobné projekty, pro velké projekty je z důvodu dlouhého čekání na vývoj nevhodný. [35]

1.5.3 Systémové testování

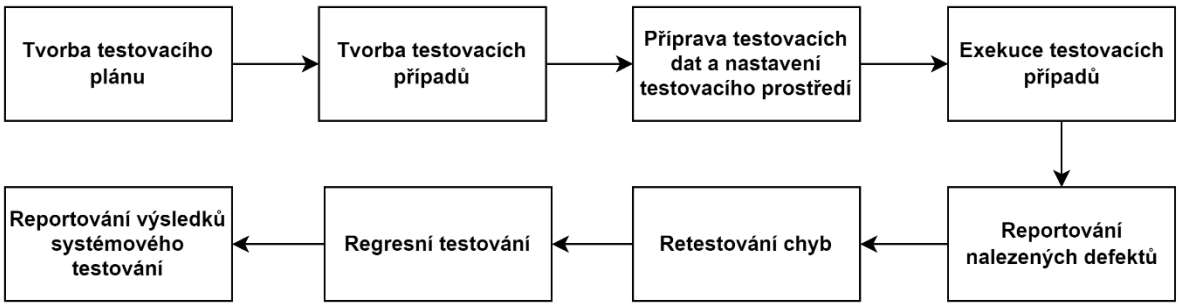
Systémové testování navazuje na integrační testování a porovnává funkčnost vytvořeného softwaru jako celek s požadavky specifikovanými v systémové specifikaci. Ve srovnání s předchozími úrovněmi je testování zaměřeno na systém jako celek a ověřuje především jeho funkčnost. Zatímco například v rámci integračního testování jsou testovány integrace mezi jednotlivými moduly aplikace, během systémového testování se testují všechny moduly dohromady E2E. Lze konstatovat, že systémové testování zajišťuje správnost a kompletnost softwaru. [36]

Využívá techniky black-box testování, hodnotí funkčnost systému větší měrou z pohledu uživatele a nevyžaduje znalosti vnitřní struktury aplikací a kódu. Systémové testování je zaměřeno především na externí rozhraní, komplexní vlastnosti, bezpečnost, výkon a použitelnost aplikace. Mělo by být prováděno na testovacím prostředí, které se shoduje v maximální možné míře s prostředím produkčním. Úspěšnost testování z velké části závisí na kvalitě dodané specifikace, která by měla obsahovat jasně popsané požadavky, dle kterých může tester danou aplikaci testovat. [37]

Výstupní kritéria systémového testování:

- Proběhla exekuce všech testovacích případů.
- V průběhu exekuce nebyly nalezeny žádné kritické chyby. Pokud chyby nalezeny byly, byly následně opraveny a retestovány.
- Report o výsledcích testování byl předán zainteresovaným osobám. [37]

Na obrázku č. 14 jsou zobrazeny všechny fáze systémového testování dle pořadí, ve kterém jsou vykonávány.



Obrázek 14 – Průběh systémového testování

Zdroj: vlastní zpracování v programu draw.io dle [37]

1.5.4 Akceptační testování

Akceptační testování (mnohdy označované jako UAT testování) je zpravidla poslední úrovní testování, ve které se software neporovnává se zadanou specifikací, ale s očekáváním koncových uživatelů. Cílem akceptačního testování je dokázat, že software dostatečně splňuje potřeby uživatelů, což umožňuje jeho nasazení do živého prostředí. Může dojít k situaci, kdy dodávaný produkt projde testy ve všech úrovních testování, a až akceptační testy zjistí, že nesplňuje potřeby uživatelů, a tudíž je pro ně nepřijatelný. [36]

Software by měl být vždy použitelný a jeho užíváním by neměla vznikat vyšší míra nespokojenosti mezi uživateli. Použitelný produkt je takový, se kterým se uživatelé naučí rychle a efektivně pracovat. Použitelnost zahrnuje kromě subjektivní spokojenosti uživatelů také objektivní kritéria, například počet vzniklých chyb, dobu trvání a počet operací, které musí být provedeny v průběhu interakce se systémem za účelem dokončení požadovaného úkolu. [36]

Pro provedení akceptačních testů je třeba vybrat vhodné kandidáty z řad uživatelů, kteří mají zkušenosti s dosavadními systémy a mají povědomí o stávajících procesech. Testeři plánují a organizují, co, jak a kdy má být otestováno.

Dobře zorganizované akceptační testy přinesou objektivní výsledky, které lze vyhodnotit a na jejichž základě může být rozhodnuto, zda vyvinutý produkt může být nasazen na produkční prostředí, nebo je třeba provést změny dle nalezených problémů. Zde je důležité zmínit, že chyby nalezené v této úrovni testování jsou nejnákladnější na opravu, nákladnější je již pouze oprava defektů nalezeného na produkčním prostředí. [36]

2 FORMULACE PROBLÉMU

Cílem diplomové práce je vytvoření manuálu zabývajícího se API testováním s využitím nástroje Postman. V následující kapitole se proto zaměřím na popis application programming interface, na nástroj Postman, na příklad testování API a na manuál k aplikaci Postman.

Úvod třetí kapitoly bude věnován HTTP protokolu, jeho metodám a stavovým kódům. Součástí kapitoly bude charakteristika nejčastěji používaných API, která jsou využívána pro komunikaci mezi aplikacemi. Dále bude představen a popsán nástroj Postman, přičemž budou zmíněny také alternativní nástroje použitelné pro testování API.

V další části bude na příkladu znázorněno testování API s využitím testovacího nástroje Postman. Pro testování bude vybráno veřejně dostupné API. Testování si klade za úkol zjistit, zda API poskytuje správné informace v souladu s dokumentací API. Dále zda API správně vrací formát a obsah výstupních dat a vrácené hodnoty stavových kódů odpovídají očekávaným hodnotám. Testování API bude prováděno s využitím funkcí, které poskytuje nástroj Postman.

V závěru třetí kapitoly bude popsán návod k aplikaci Postman, který bude vytvořen za účelem, aby sloužil jako příručka pro využití tohoto nástroje k testování API. V poslední kapitole diplomové práce budou zhodnoceny dosažené výsledky.

3 INTEGRAČNÍ TESTOVÁNÍ S VYUŽITÍM NÁSTROJE POSTMAN

Postman je nástroj, který uživatelům umožňuje testovat, vyvíjet a dokumentovat API. Postman umožňuje uživatelům konfigurovat různé aspekty požadavku, jako jsou hlavičky, tělo požadavku, parametry URL a zobrazuje odpověď včetně HTTP kódu a hlaviček.

Obvykle jej používají vývojáři a testeři pro testování, ladění a dokumentování API. Disponuje uživatelsky přívětivým rozhraním, které uživatelům umožňuje snadno provádět různé HTTP dotazovací metody, jako jsou GET, POST, PUT a DELETE. Umožňuje uživatelům ukládat a organizovat požadavky do kolekcí požadavků. S využitím správného nastavení proměnných prostředí lze jednoduše použít na různých vývojových prostředích.

3.1 HTTP protokol

HTTP protokol neboli Hypertext Transfer Protocol, slouží jako komunikační protokol pro přenos dat mezi webovým prohlížečem a servery. Tento protokol funguje na základě klient-server architektury. Klient (například webový prohlížeč) odesílá požadavek na server v podobě HTTP requestu a server na něj následně odpovídá v podobě HTTP response. Pomocí HTTP protokolu lze specifikovat jednoznačné umístění zdroje v internetové síti díky URL (Uniform Resource Locator). HTTP protokol je užíván pro webové služby a je základem většiny internetových aplikací. V dnešní době je již standardem zabezpečená verze HTTPS (Hypertext Transfer Protocol Secure), která používá protokol SSL nebo TLS pro šifrování dat. [38]

3.1.1 Metody HTTP požadavků

HTTP požadavek vlastní metodu definující typ akce, kterou se uživatel chystá provést. Mezi nejčastěji používané HTTP metody patří: [39]

- **GET** – Metoda GET je nejpoužívanější metodou, která slouží k získání informací ze serveru.
- **POST** – Metoda POST se používá k odesílání uživatelských dat na server, kde jsou data zpracována a uložena.
- **PUT** – Metoda PUT se používá k vytváření nebo k aktualizaci zdrojů na serveru. Klient pomocí metody PUT pošle požadavek na server společně s daty, která chce na server uložit. Server poté ověřuje, zda má klient oprávnění pro aktualizaci nebo pro vytvoření zdroje. V případě, že klient vlastní oprávnění, data se uloží na server.

- **DELETE –** Metoda DELETE je používána k odstranění zdroje na straně serveru. Je nevratná, odstraněný zdroj není možné obnovit.
- **PATCH** – Metoda PATCH je používána k aktualizaci existujícího zdroje na serveru. Oproti metodě PUT požadavek PATCH nevyžaduje, aby klient poslal celý obsah zdroje. Je často používán k aktualizaci velkých zdrojů, kdy jsou zaslány jen specifické části zdroje.
- **OPTIONS** – Tato metoda se používá k získání informací o podporovaných HTTP metodách na serveru.
- **HEAD** – Metoda HEAD je blízká metodě GET. Používá se k získání hlaviček odpovědi od serveru, aniž by se vracel celý obsah zdroje.

Výše zmíněné metody nejsou jediné, existuje spousta dalších, například metoda CONNECT nebo metoda Trace. Metody HEAD, OPTIONS, GET, TRACE jsou bezpečné metody, tj. vícekrát zaslaný požadavek na stejný zdroj by měl vrátit stejný výsledek a nemělo by dojít k žádným změnám na straně serveru. [39]

3.1.2 HTTP stavové kódy

HTTP response se skládá ze stavového kódu a těla odpovědi, které nese přenášená data. Stavové kódy představují kódy, které server posílá klientovi jako odpověď na jeho HTTP požadavek. Kód indikuje výsledek zpracování požadavku. [40]

Stavové kódy jsou tvořeny třemi číslicemi a dle logických podobností se rozdělují do 5 skupin:

- **1xx – Informativní kódy** – Kódy začínající číslicí 1xx jsou informativní kódy, které oznamují, že:
    - server přijal a pochopil požadavek klienta.
    - server nadále zpracovává požadavek.

  Například kód 100 - Continue. [40]
- **2xx** – **Kódy pro úspěšné požadavky** – HTTP stavové kódy v kategorii 2xx indikují, že klientův požadavek byl úspěšně zpracován. Oznamují, že vše proběhlo v pořádku a další akce ze strany klienta nejsou nutné. Mezi nejznámější kódy v této skupině patří například 200 – OK, který indikuje, že požadavek byl úspěšně zpracován, 201 – Created, který oznamuje, že byl vytvořen nový zdroj, nebo 204 – No Content, který znamená, že server úspěšně vykonal požadavek, ale nevrátil žádná data. [40]

- **3xx – Kódy pro přesměrování požadavku** – HTTP stavové kódy v kategorii 3xx oznamují, že požadavek byl úspěšně zpracován, ale klient musí provést další akci k dokončení požadavku. Jsou obvykle používány k přesměrování URL na jinou adresu. Například 301 - Moved Permanently nebo 307 - Temporary Redirect. [40]
- **4xx – Kódy pro chyby klienta** – Oznamují, že požadavek obsahuje chybu a nelze jej proto zpracovat. Jedná se například o 400 – Bad Request, 401 – Unauthorized, 403 – Forbidden, 408 – Request Timeout. [40]
- **5xx – Kódy pro chyby serveru** – Oznamují, že požadavek nebyl úspěšně zpracován kvůli chybě na straně serveru. Chyba může být způsobena mimo jiné nedostupností serveru nebo chybou v aplikaci, která server provozuje. Např. 500 – Internal Server Error, 503 – Service Unavailable, 504 – Gateway Timeout. [40]

3.2 API

API (Application Programming Interface) je rozhraní, které umožňuje komunikaci a sdílení dat mezi aplikacemi. Rozhraní API jsou různá, tato práce se zaměřuje především na webová API.

API založené na HTTP se často nazývají webová API, neboť se používají pro přístup k webovým aplikacím. Aplikace, ke kterým se přistupuje přes webová API, se často nazývají webové služby. [41]

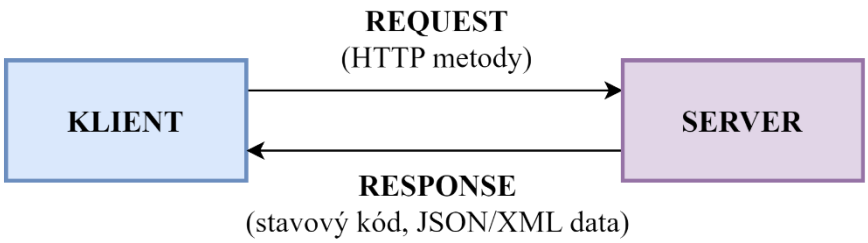
V následujících podkapitolách jsou popsány nejpoužívanější API dle architektury.

3.2.1 REST API

Rest (Representational State Transfer) poskytuje přístup k datům pomocí HTTP protokolu a standardních HTTP metod (GET, POST, PUT, DELETE). Rest byl poprvé definován v roce 2000 Royem Fieldingem na University of California v jeho disertační práci, ve které se zabýval architekturou webu.

Rest poskytuje standardizovaný způsob pro komunikaci mezi klientem a serverem. Klient odesílá HTTP požadavek na server s určitou URL, která identifikuje zdroj dat, který chce klient získat nebo upravit. Server poté reaguje na požadavek poskytnutím odpovědi v podobě dat nebo chybové zprávy. REST používá HTTP pro všechny CRUD operace (Create = POST, Read = GET, Update = PUT, Delete = DELETE). [42]

Na obrázku č. 15 je znázorněn REST API model.



Obrázek 15 – REST API model

Zdroj: vlastní zpracování v programu draw.io

REST API používá URI (Uniform Resource Identifier) pro identifikaci zdroje. URI se skládá z adresy serveru a cesty ke zdroji. Například http://www.test.cz/data/soubor.txt.

Základními vlastnostmi REST API jsou:

- **Beze-stavovost** – server neukládá informace o klientovi. Každý request musí obsahovat všechny informace potřebné k jeho provedení.
- **Klient-server architektura**.
- **Kešovatelnost** – klient si může uložit data a následně je využívat bez nutnosti opakovaného stahování ze serveru.
- **Systém je rozdělen do několika vrstev** – jednotlivé vrstvy poskytují různé funkce, například funkce autentizace, autorizace nebo šifrování.

Nejčastěji využívaným formátem dat pro komunikaci mezi klientem a serverem je JSON, ale existují i další formáty dat, které lze využít (např. XML, CSV apod.). Z jeho vlastností plyne nevýhoda restu, kterou je overfetching (přebytečné načítání dat) nebo underfetching (nedostatečné načítání dat). [42]

3.2.2 SOAP API

SOAP (Simple Object Access Protocol) API je typ webového API, který se zaměřuje na poskytování datových služeb prostřednictvím standardizovaného formátu zpráv SOAP.

SOAP zpráva se skládá z obálky <soap:Envelope>, z hlavičky <soap:Header> (volitelné) a z těla (<soap:Body>). Každý <soap:Envelope> element musí obsahovat přesně jeden <soap:Body> element a maximálně jeden <soap:Header> element. V <soap:Body> se nachází obsah zprávy určený k přenosu. [43]

Struktura jednoduché SOAP zprávy je zobrazena na obrázku č. 16.

```xml
<?xml version="1.0"?>
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope/">
  <soap:Header>
  ...
  </soap:Header>
  <soap:Body>
  ...
  </soap:Body>
</soap:Envelope>
```

Obrázek 16 – Struktura SOAP zprávy

Zdroj: vlastní zpracování v programu Visual Studio Code

SOAP je protokol založený na XML, který se používá pro komunikaci mezi aplikacemi pomocí HTTP nebo HTTPS protokolu. SOAP je nezávislý na platformě a je přenosný. Nevýhodou je jeho komplexita a mnoho informací v SOAP a WSDL, které jsou nadbytečné. Dochází tak ke zvýšení objemu síťové komunikace a k většímu zatížení serveru. [44]

Protokol SOAP se využívá v kombinaci s technologiemi WSDL a UDDI. Jazyk WSDL slouží k popisu webových služeb a byl vytvořen společným úsilím firem Microsoft a IBM s cílem sjednotit jazyk používaný pro popis rozhraní webových služeb. WSDL se používá k definování rozhraní pro webové služby, včetně operací, které webové služby poskytují. [45]

Registr UDDI slouží jako katalog webových služeb, který umožňuje aplikacím vyhledávat a nalézat webové služby pomocí kategorií a klíčových slov. SOAP se používá pro komunikaci mezi aplikacemi, WSDL k definování rozhraní webové služby a UDDI k registraci nebo k vyhledávání webových služeb. Tyto tři technologie dohromady tvoří základ infrastruktury webových služeb. [45]

3.2.3 GraphQL API

GraphQL API poskytuje flexibilní způsob dotazování na data pomocí jazyka GraphQL. GraphQL bylo vytvořeno společností Facebook v roce 2012 v souvislosti s vývojem mobilní aplikace Facebook za účelem odstranění nedostatků REST API. Od roku 2015 je k dispozici jako open source. [46]

GraphQL je dotazovací jazyk, který umožňuje klientovi definovat požadavek na data, která požaduje, a po vrácení odpovědi obdrží přesně taková data, která požadoval, a žádná další data navíc. Touto vlastností se zásadně liší od REST API, při jehož použití je nutné vždy stáhnout celý obsah dat, a to i v případě, že klienta zajímá pouze konkrétní část dat. [46]

Pro načítání dat ze serveru se používá dotaz (query), pro nahrávání dat, aktualizaci dat a mazání dat jsou používány mutace (mutation). Pro definici API slouží typový systém, jenž tvoří schéma a definuje možnosti přístupu k datům. Schéma je tvořené jazykem GraphQL SDL (schema definition language). [46]

U GraphQL API jeden koncový bod zpravidla obsahuje všechny informace. Důležité je znát schéma nebo strukturu dat, aby se uživatel mohl efektivně dotazovat. [47]

Stručný popis komunikace klienta se serverem:

1. Klient definuje požadavek na data v dotazu pomocí GraphQL syntaxe.
2. Server daný požadavek přijme a ověří jeho syntaktickou správnost.
3. Server zpracuje požadavek a vrátí pouze požadovaná data v podobě JSON objektu.
4. Klient obdrží odpověď. [46]

Výhody použití GraphQL:

- Rychlost načítání, efektivita.
- Vhodné pro komplexní systémy a mikro služby.
- Hierarchická struktura.
- Automatická dokumentace.
- Snadná testovatelnost manuálně i automatizovaně. [46]

Nevýhody použití GraphQL:

- Komplexní a dlouhé dotazy.
- Složitější údržba ve srovnání s REST API.
- Omezenější podpora pro integraci s jinými technologiemi.
- Cachování – složitější ukládání do mezi paměti, jelikož každý dotaz může být jiný.
- Obtížnější omezení jednotlivých typů dotazů. [46]

3.2.4 gRPC API

gRPC je novější a výkonnější implementace protokolu RPC. RPC (Remote Procedure Call) je protokol pro volání procedur ze vzdáleného serveru. Jedná se o open-source framework pro vývoj a implementaci distribuovaných aplikací. gRPC používá protokol RPC a rozšiřuje

jej o další funkce, například o podporu více programovacích jazyků, obousměrné streamování a šifrování dat a o kompresi dat. Hlavní rozdíl mezi RPC a gRPC je výkonnost a funkcionalita. gRPC nabízí výkonnější a funkčně bohatší protokol pro komunikaci mezi aplikacemi a službami. [48]

gRPC je na platformně nezávislý, podporuje mimo jiné jazyky C++, Java, Python, Go, Ruby. Využívá binární formát pro komunikaci mezi klientem a serverem, data jsou přenášena rychle, a proto je využíván pro vysoce výkonné služby a mikro služby. gRPC podporuje použití TLS a autentizaci na základě tokenů. Využívá Protokol Buffer jako jazyk popisující rozhraní (IDL) přes HTTP/2 protokol. [48]

3.3 Postman

Postman je nástroj pro správu a testování API. Vlastníkem aplikace Postman je firma Postman Inc., kterou založil v roce 2012 současný CEO Abhinav Asthana. Hlavní sídla firmy se nacházejí v San Franciscu a v Bangalore. V současnosti ve firmě pracuje zhruba 500 až 1000 zaměstnanců. V roce 2023 Postman používá přes 25 milionů uživatelů. [49]

Postman může být použit k návrhu API a k následné validaci návrhu, že API splňuje zadané požadavky. V průběhu vývoje umožňuje vývojářům a testerům manuální i automatizované testování API. Postman je možné integrovat s dalším softwarem. Může být také použit ke tvorbě a správě dokumentace API nebo k monitorování API v reálném čase.

3.3.1 API klient

Postman disponuje intuitivním a přehledným rozhraním. Rozhraní Postmanu se skládá z následujících částí:

- **Header** (hlavička),
- **Workbench** (pracovní stůl),
- **Left sidebar** (levý postranní panel),
- **Right sidebar** (pravý postranní panel),
- **Footer** (patička).

Hlavička obsahuje záložky Home, Workspaces, API Network a Explore.

Záložka Home je výchozím bodem pro práci s aplikací. Poskytuje přehled o kolekcích, historii požadavků a nedávných aktivitách. Slouží jako rozcestník pro přesun do dalších částí aplikace (vytvoření nového požadavku, vytvoření nové kolekce nebo spuštění testů).

Záložka Workspaces je určena k organizaci a řízení pracovních prostorů. V rámci Workspaces je možné oddělit projekty do samostatných pracovních prostorů a přidávat do nich konkrétní kolekce požadavků či spravovat dokumentaci. Pro jednotlivé pracovní prostory lze nastavit různá přístupová oprávnění pro jednotlivé členy týmu. Workbench, Left sidebar, Right sidebar a Footer jsou dostupné ze záložky Workspaces.

Workbench je určen pro vývoj, testování nebo ladění API a umožňuje vývojářům vytvářet a testovat API požadavky a odpovědi bez nutnosti spouštět celou aplikaci. Poskytuje funkce pro automatické generování testovacího kódu a nástroje pro vyhodnocování odpovědí API.

Left sidebar obsahuje rozcestník následujících položek:

- *Collections* – slouží ke správě a organizaci requestů.
- *Environments* – záložka je určena pro správu prostředí a nastavování hodnot proměnných.
- *Mock Servers* – slouží k vytvoření simulovaných odpovědí API, které umožňují testování bez nutnosti komunikace se skutečnými API.
- *Monitors* – monitorování poskytuje informace o stavu API. V případě identifikace problémů jsou konkrétním uživatelům zasílána upozornění.
- *Flows* – měla by umožnit automatizaci série požadavků, jedná se o plánovanou funkci, která je v momentálně nejnovější verzi 10.9.4 aplikace Postman nedostupná.
- *History* – záložka zobrazuje historii požadavků na API, a to včetně odpovědí serveru.

Right sidebar poskytuje nástroje k získání základních informací o API, k dokumentaci, ke komentářům nebo ke generování kódu v různých jazycích. Footer zobrazuje nástroje Find and Replace, Console, Cookies, Capture requests, Runner a Trash.

Záložka API Network slouží k publikování a správě API dokumentace.

V záložce Explore může uživatel vyhledávat API, které byly publikovány v rámci Postman API Network a studovat jejich dokumentaci. Vyhledávání je možné podle několika kritérií, mimo jiné podle názvu API, podle popisu API nebo podle autorů. Záložka Explore může sloužit také jako zdroj inspirace pro vývoj vlastního API případně pro integraci existujícího API. [50]

3.3.2 Tarify aplikace Postman

Z hlediska cenových tarifů je možné vybrat ze 4 různých plánů – Free, Basic, Professional a Enterprise.

Postman lze ve verzi Free používat bezplatně s jeho využitím až pro 3 uživatele v týmu. S každým upgradem na vyšší plán lze vždy rozšířit Postman o další funkce nebo zvýšit omezení daná aktuálně používanou verzí dle potřeb projektu. [51]

V tabulce 2 jsou zobrazeny rozdíly mezi jednotlivými tarify. Z tabulky byly vyřazeny vlastnosti, které jsou identické pro všechny tarify.

Tabulka 2 – Odlišnosti mezi jednotlivými tarify aplikace Postman

Vlastnosti	Free	Basic	Professional	Enterprise
Collection Runner (počet běhů za měsíc)	25	25	250	Unlimited
Volání Postman API (měsíčně)	1	10	100	1,000,000
Celkový počet integrací	5	10	50	100
Nativní podpora Git	Ne	Ne	Ano	Ano
Statické IP adresy pro monitorování API	Ne	Ne	Ano	Ano
Počet vlastních domén	Ne	1	5	25
Private workspaces	Ne	Ne	Ano	Ano
Partner workspaces, Private API Network	Ne	Ne	Ne	Ano
Reporting a analytika	Ne	Ne	Ne	Ano
Role a oprávnění	Ne	Ne	Ano	Ano
Jednotné přihlašování (SSO)	Ne	Ne	Google Workspaces	SSO Providers and SAML
Automatická správa uživatelů, skupiny uživatelů a	Ne	Ne	Ne	Ano

Vlastnosti	Free	Basic	Professional	Enterprise
zachycení domény				
Obnova smazaných kolekcí	1 den	30 dní	90 dní	90 dní
Přístup k uživatelským logům, API logům	Ne	Ne	Ano	Ano
Role super admina	Ne	Ne	Ne	Ano
Řízení nasazení	Ne	Ne	Ne	Ano
Tvůrce vlastních pravidel	Ne	Ne	Ne	Ano
Mechanismus pro ověřování požadavků a schémat	Ne	Ne	Ne	Ano
Bezpečnostní varování	Ne	Ne	Ne	Ano

Zdroj: vlastní zpracování v programu Microsoft Excel dle [51]

V tabulce 3 jsou znázorněny ceny jednotlivých tarifů, které si mohou jednotlivci nebo organizace zvolit. Cena je vždy účtována měsíčně za každého uživatele.

Tabulka 3 – Cenové porovnání jednotlivých tarifů

Tarif	Cena při měsíční platbě	Cena při roční platbě
Free	Zdarma	Zdarma
Basic	15 dolarů	12 dolarů
Professional	36 dolarů	29 dolarů
Enterprise	Tato varianta není k dispozici.	99 dolarů

Zdroj: vlastní zpracování v programu Microsoft Excel dle [51]

3.3.3 Další nástroje pro testování API

Postman není jediným nástrojem k testování API. Na trhu je dostupný také další software, který je možné využít. Tato kapitola se zabývá nástroji Insomnia, SoapUI a Apache JMeter™.

Insomnia

Insomnia je open-source software vhodný pro odesílání požadavků REST, SOAP, GraphQL a gRPC. Poskytuje přívětivé uživatelské prostředí v rámci minimalistické desktopové aplikace. Podporuje práci s proměnnými, s collections, s prostředími, s workspaces a také umožňuje posílat požadavky s různými HTTP metodami. Ve srovnání s Postmanem neobsahuje funkce monitoringu a generování dokumentace API. [52]

Nevýhodou může být menší internetová uživatelská komunita. Naopak umožňuje integraci s dalšími nástroji prostřednictvím pluginů, které si uživatel může jednoduše nainstalovat. Tyto pluginy mohou uživatelé sami vytvářet a případně sdílet s ostatními. V případě, že chce tester upravovat předpřipravené testovací skripty v Insomnii, nebude mu stačit free verze. Insomnia umožňuje historizaci přijatých odpovědí dle jednotlivých requestů. [52]

Postman i Insomnia se v současnosti řadí mezi nejpoužívanější nástroje k testování API. Postman je komplexnější, nicméně oba nástroje disponují obdobnými funkcemi, proto rozhodování pro výběr mezi nimi může být často subjektivní. [52]

SoapUI

SoapUI je open-source nástroj pro testování webových služeb, který poskytuje uživatelské rozhraní pro vytváření, odesílání a testování SOAP a REST požadavků. SoapUI poskytuje funkce pro automatizaci testů, správu projektů a integraci s dalšími nástroji pro testování. Ve srovnání s nástrojem Postman je SoapUI vhodnější pro nefunkční testování (bezpečnostní testování, zátěžové testování). [53]

Testování se SoapUI není omezeno pouze na API. Je větší měrou zaměřeno na SOAP, zatímco Postman je zaměřen více na REST. Postman pro psaní testovacích skriptů používá programovací jazyk Javasript, zatímco v SoapUI lze skripty psát s pomocí Groovy. Postman disponuje moderním uživatelským rozhraním, naproti tomu SoapUI poskytuje více komplexnějších funkcí. [53]

Apache JMeter™

Apache JMeter™ je open-source software vyvinutý společností Apache Software Foundation. Hlavní využití nachází v oblasti testování výkonu. Nevýhodou je, že JMeter podporuje výlučně operační systémy Windows. Poskytuje nástroje pro analýzu, nahrávání a skriptování. Nástroj byl původně vytvořený primárně pro zátěžové testování, ale v současnosti je používán také pro

funkční testování API. Komplexnější testování API je možné, ale ve srovnání s Postmanem komplikovanější. [54]

Všechny výše zmíněné nástroje mají své výhody a nevýhody. Je příhodné vždy rozlišovat podle běžícího projektu a vybrat takový nástroj, který odpovídá projektovým potřebám. Není nezbytné používat výlučně jeden nástroj, ale je možné je vzájemně kombinovat a vytěžit tím maximum z jimi nabízených funkcí. [54]

3.4 Příklad na testování API

V této kapitole bude předvedeno na vybraném příkladu testování API. Jako základní kritérium pro výběr vhodného API bylo stanoveno, že API musí obsahovat dokumentaci popisující:

- jaké endpointy jsou k dispozici,
- jaké metody lze použít,
- jaké jsou očekávané vstupy a výstupy.

Podmínkou je, že API musí být veřejně dostupné.

Z tohoto důvodu bylo na základě stanovených kritérií vybráno veřejně dostupné API České pošty. Testování se bude zaměřovat pouze na službu Address.

3.4.1 Popis zvoleného API

API České pošty poskytuje informace o adresách, o historii zásilek, o poštovních směrovacích číslech a o poštách. Rozhraní REST je popsáno formou automatizovaně zpracovatelných WADL (Web Application Description Language) souborů. V těchto souborech je popsáno rozhraní, zdroje rozhraní, metody a parametry, které lze v requestu použít.

Veškerá dokumentace API je dostupná na webové adrese https://b2c.cpost.cz. [55]

V tabulce 4 je sepsán seznam služeb s jejich popisem, které API poskytuje.

Tabulka 4 – Popis služeb

Název služby	Popis
Address	Služba poskytuje identifikátor adresy postupným výběrem kraje, okresu, obce, části obce, ulice a čísla. Identifikátor adresy slouží jako vstupní parametr pro poskytnutí informace o odpoledním doručování.
Parcel history	Služba poskytuje historii stavů požadované zásilky.
Post code	Služba poskytuje PSČ adresy a informaci, zda je v rámci PSČ poskytováno doručování v časových pásmech.
Post office information	Služba poskytuje informace o poštách.
Afternoon parcel delivery	Služba poskytuje informace o odpoledním doručování pro požadovanou adresu.
Parcel rating	Služba poskytuje zadání spokojenosti s podáním a dodáním dané zásilky.

Zdroj: vlastní zpracování dle [55]

V tabulce 5 je zobrazen seznam služeb s odkazy na soubory WADL.

Tabulka 5 – Soubory WADL ke službám

Název služby	URL
Address	b2c.cpost.cz/services/Address/application.wadl
Parcel history	b2c.cpost.cz/services/ParcelHistory/application.wadl
Post code	b2c.cpost.cz/services/PostCode/application.wadl
Post office information	b2c.cpost.cz/services/PostOfficeInformation/application.wadl
Afternoon parcel delivery	b2c.cpost.cz/services/AfternoonParcelDelivery/application.wadl
Parcel rating	b2c.cpost.cz/services/ParcelRating/application.wadl

Zdroj: vlastní zpracování dle [55]

3.4.2 Návrh testovacích případů

Základním zdrojem informací pro návrh testovacích případů je API dokumentace, ve které je definováno, jak by se konkrétní služba měla chovat a jaké stavové kódy by měla vracet.

Bude provedeno funkční testování, testování vstupních dat a testování výstupních dat. V rámci testování budou provedeny pozitivní i negativní testy.

Pozitivní testy ověřují, že rozhraní API přijímá validní vstupy, vrací očekávané vstupy a vrací odpovídající stavové kódy. Negativními testy bude ověřeno, že rozhraní API vrací správnou odpověď, když očekávaný výstup neexistuje nebo při přijímání neplatných vstupů.

Vzhledem k tomu, že se jedná o API v živém provozu, nebudou prováděny bezpečnostní ani zátěžové testy. Ze stejného důvodu není možné otestovat stavový kód 503. API je veřejně dostupná a nevyžaduje autentizaci.

Níže je zobrazen návrh testovacích případů pro getRegionListAsJson.

Navržené testy pro getDistrictListAsJson, getCityListAsJson, getCityPartListAsJson, getStreetListAsJson a getNumberListAsJson jsou uvedeny v příloze (Příloha A .

1) Návrh testovacích případů pro getRegionList

POZITIVNÍ TESTY:

a) Test, který ověří, že vrácená odpověď obsahuje pole, které není prázdné.

b) Test, který ověří, že každý objekt v poli obsahuje pouze vlastnosti id a name.

c) Test, který ověří, že hodnoty vlastností id jsou datového typu string.

d) Test, který ověří, že hodnota vlastnosti name v každém objektu obsahuje pouze písmena, přičemž první písmeno je vždy velké.

e) Test, který ověří, že tělo odpovědi obsahuje 14 objektů.

f) Test, který ověří, že hodnota stavového kódu při validním vstupu je 200.

g) Test, který ověří, že API nevrací duplicitní hodnoty.

h) Test, který ověří, že Address/getRegionListAsXml vrací stejné hodnoty jako Address/getRegionListAsJson.

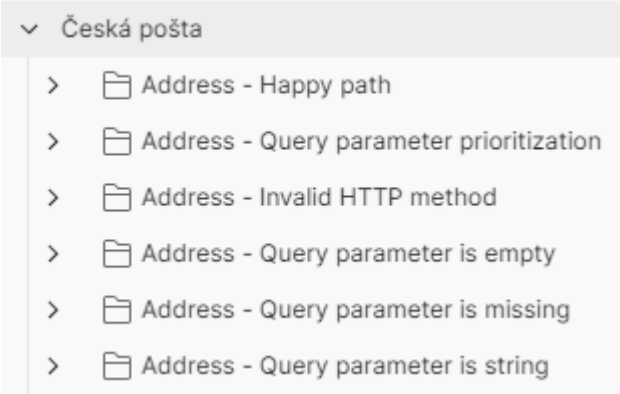
NEGATIVNÍ TESTY:

i) Test, který ověří, že při použití HTTP metody POST bude vrácen stavový kód 405.

3.4.3 Implementace testovacích případů

Implementované testovací skripty byly vytvořeny dle navržených testovacích případů v kapitole 3.5.2.

Testy byly zorganizovány v rámci collection **Česká pošta** do 6 složek. Složky **Address – Happy path** a **Address - Query parameter prioritization** obsahují pozitivní testy. Zbylé složky obsahují testy negativní. Struktura s názvy složek je zobrazena na obrázku 17.



Obrázek 17 – Organizace do složek

Zdroj: vlastní zpracování v aplikaci Postman [56]

Níže je zobrazen implementovaný testovací skript pro getRegionListAsJson.

Další implementované testovací skripty pro služby getDistrictListAsJson, getCityListAsJson, getCityPartListAsJson, getStreetListAsJson a getNumberListAsJson jsou uvedeny v příloze (Příloha BPříloha BPříloha B).

1) Implementované testovací skripty pro getRegionList

a) Test, který ověří, že vrácená odpověď obsahuje pole, které není prázdné.

```
pm.test("Odpověď obsahuje pole, které není prázdné", function() {
    pm.expect(jsonData).to.be.an("array").that.is.not.empty;
});
```

b) Test, který ověří, že každý objekt v poli obsahuje pouze vlastnosti id a name.

```
pm.test("Každý objekt v poli obsahuje pouze vlastnosti id a name", function() {
    jsonData.forEach(function(region) {
        pm.expect(Object.keys(region)).to.have.members(["id", "name"]);
    });
});
```

c) Test, který ověří, že hodnoty vlastností id jsou datového typu string.

```
pm.test("Hodnoty vlastností id jsou datového typu string", function() {
    jsonData.forEach(function(region) {
        pm.expect(region.name).to.be.a("string").that.is.not.empty;
    });
});
```

d) Test, který ověří, že hodnota vlastnosti name v každém objektu obsahuje pouze písmena, přičemž první písmeno je vždy velké.

```
pm.test("Hodnota vlastnosti name v každém objektu obsahuje pouze písmena a první
písmeno je vždy velké", function() {
    jsonData.forEach(function(region) {
        pm.expect(region.name).to.match(/^[A-ZŠČŘŽÝÁÍÉÚŮĚÔŇŤ][\p{L}\s-]*$/u);
    });
});
```

e) Test, který ověří, že tělo odpovědi obsahuje 14 objektů.

```
pm.test("Tělo odpovědi obsahuje 14 objektů", function() {
    jsonData.forEach(function(region) {
        pm.expect(jsonData).to.have.lengthOf(14);
    });
});
```

f) Test, který ověří, že hodnota stavového kódu při validním vstupu je 200.

```
pm.test("Stavový kód má hodnotu 200", function () {
    pm.response.to.have.status(200);});
```

g) Test, který ověří, že API nevrací duplicitní hodnoty.

```
pm.test("API nevrací duplicitní hodnoty", function() {
    let ids = jsonData.map(function(idMapper) {
        return idMapper.id;
    });
    pm.expect(ids.length).to.equal(new Set(ids).size);
});
```

h) Test, který ověří, že Address/getRegionListAsXml vrací stejné hodnoty jako Address/getRegionListAsJson.

```
let responseJson = xml2Json(responseBody);

let regions = [];
if (Array.isArray(responseJson.regions.region)) {
    regions = responseJson.regions.region.map(region => {
        return {
            id: region.id,
            name: region.name
        };
    });
```

```
} else {
    regions = [{
        id: responseJson.regions.region.id,
        name: responseJson.regions.region.name
    }];
}

let classicJson = JSON.parse(pm.collectionVariables.get("regionJson"));

pm.test("XML odpověď vrací stejné hodnoty jako JSON odpověď.", function() {
    pm.expect(regions).to.eql(classicJson);
});
```

i) Test, který ověří, že při použití HTTP metody POST bude vrácen stavový kód 405.

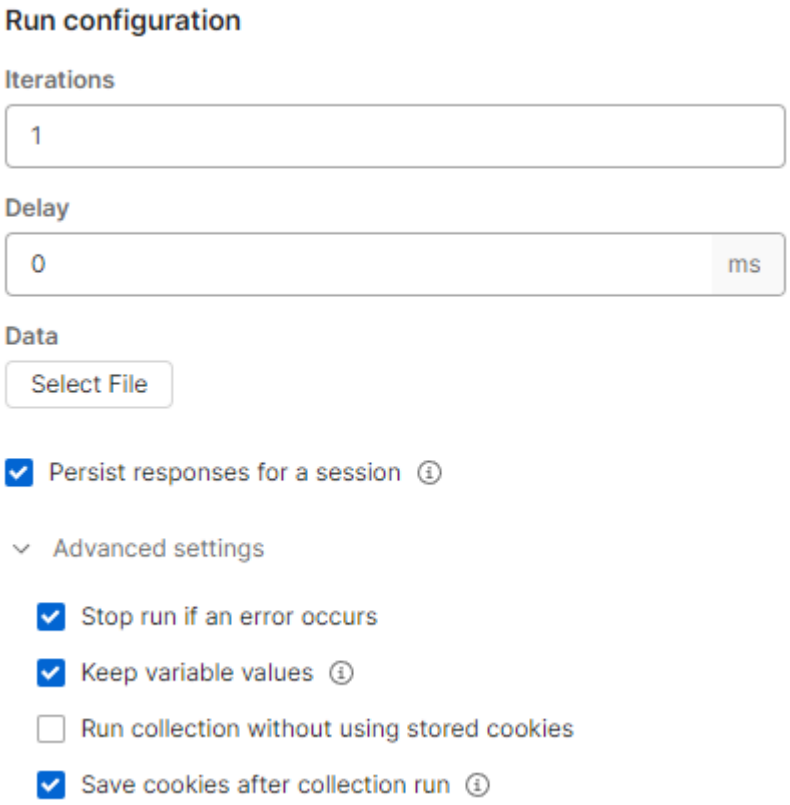
```
pm.test("Stavový kód má hodnotu 405", function() {

    pm.expect(pm.response.code).to.eql(405);
});
```

3.4.4 Výsledky exekuce testovacích případů

Exekuce byla provedena s využitím nástroje Collection Runner. Běh byl spuštěn manuálně.

Konfigurace Collection Runneru před spuštěním exekuce testů je zobrazena na obrázku 18.



Obrázek 18 – Nastavené parametry v Collection Runneru

Zdroj: vlastní zpracování v aplikaci Postman [56]

Jak uvádí obrázek 19, bylo provedeno 101 testů, z nichž 93 bylo vyhodnoceno jako passed a 8 jako failed.

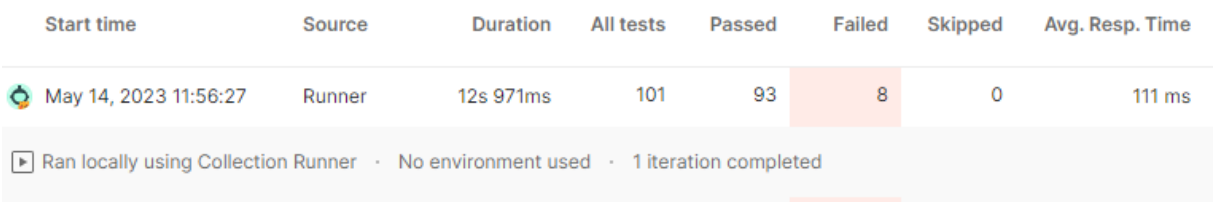
$$\text{Procentuální vyjádření testů vyhodnocených jako passed} = \frac{\text{Počet passed testů}}{\text{Celkový počet provedených testů}} * 100$$

$$\text{Procentuální vyjádření testů vyhodnocených jako passed} = \frac{93}{101} * 100 = 92{,}08\ \%$$

$$\text{Procentuální vyjádření testů vyhodnocených jako failed} = \frac{\text{Počet failed testů}}{\text{Celkový počet provedených testů}} * 100$$

$$\text{Procentuální vyjádření testů vyhodnocených jako failed} = \frac{8}{101} * 100 = 7{,}92\ \%$$

Výsledky exekuce testů zobrazené v aplikaci Postman jsou zobrazeny na obrázku 19.



Obrázek 19 – Výsledky exekuce

Zdroj: vlastní zpracování v aplikaci Postman [56]

Všech 8 chyb bylo nalezeno u testů, které ověřují, že stavový kód je roven hodnotě 400 při odeslání requestu, který neobsahuje povinný parametr.

AssertionError, který se objevil v testu, naznačuje, že očekávaný výstup neodpovídá skutečnému výstupu, a proto test selhal. V konkrétním případě byl očekáván stavový kód 400, ale vrácen byl stavový kód 500, což není platná odpověď.

Možnou příčinou chyby tohoto typu může být špatně sestavený požadavek nebo chyba v kódu na straně serverové aplikace. Nutno dodat, že popis chování API v této situaci není v dokumentaci exaktně popsán. Dokumentace obsahuje pouze informaci, že v případě chyby se vrací v HTTP hlavičce chybový kód 400 nebo 503.

Na obrázku 20 je zobrazen detail nalezené chyby.

Obrázek 20 – Detail nalezené chyby - getDistrictListAsJson

Zdroj: vlastní zpracování v aplikaci Postman [56]

Chyba se objevila u getDistrictListAsJson, getDistrictListAsXml, getCityListAsJson, getCityListAsXml, getCityPartListAsJson, getCityPartListAsXml, getStreetListAsJson, getStreetListAsXml.

Naopak u getNumberListAsJson a getNumberListAsXml byl vrácen správně stavový kód 400.

3.5 Vytvoření manuálu k aplikaci Postman

V rámci této diplomové práce došlo k vytvoření manuálu k aplikaci Postman. Manuál slouží jako příručka pro testování a správu API. V manuálu se zaměřuji na postupné vysvětlení

jednotlivých kroků potřebných pro využívání všech důležitých funkcí, které tento nástroj nabízí.

Návod je rozdělen do 9 kapitol, které jsou doplněny o obrázky, které uživateli napomáhají s orientací v aplikaci. Vybrané kapitoly jsou doplněny o úkoly, které poskytují čtenáři prostor si vysvětlované oblasti vyzkoušet. Čtenář může díky těmto úkolům aplikovat získané znalosti z dané kapitoly samostatně v praxi a zdokonalit tak své dovednosti.

Návod předpokládá, že uživatel má základní znalosti práce s počítačem a s webovými aplikacemi. Veškeré předpoklady vhodné pro následné maximální využití funkcí nástroje Postman jsou popsány v samostatné kapitole, která je obohacena o odkazy na další studijní materiály nebo tutoriály.

Manuál podrobně popisuje, jak nainstalovat Postman na počítači a jak aplikaci spustit. Dále se věnuje přípravě a odesílání requestů a jejich organizaci v collections. Blíže popisuje možnosti exportování, importování a sdílení collections mezi více uživatelů. Návod krok po kroku vykládá, s jakými typy proměnných lze v Postmanu pracovat a jak je použít v testovacích skriptech.

Další kapitola se věnuje přípravě testovacích skriptů pro testování API, včetně testů na stavové kódy, testování hodnot v odpovědi, hromadnému spouštění testů v jedné collection a monitorování. V poslední kapitole manuálu se věnuji nástroji Newman, který umožňuje spouštět testovací skripty z příkazové řádky. Newman je velmi užitečný nástroj pro integraci funkcí softwaru Postman do CI/CD procesů.

Manuál je určen primárně pro uživatele, kteří mají velmi malé nebo žádné znalosti nástroje Postman a mají zájem se s ním naučit pracovat.

Kompletní manuál je v příloze (Příloha C) této diplomové práce.

4 ZHODNOCENÍ DOSAŽENÝCH VÝSLEDKŮ

Tato kapitola obsahuje shrnutí dosažených výsledků.

V úvodu diplomové práce byl popsán současný stav problematiky testování a zároveň byly objasněny základní pojmy související s testováním. Práce obsahuje popis aplikačního rozhraní, HTTP protokolu a nástroje Postman.

V práci bylo provedeno testování API s využitím nástroje Postman. Pro tento účel bylo vybráno API České pošty, konkrétně část API, která s pomocí HTTP metody GET poskytuje informace o adresách. V rámci práce byl popsán postup testování, který zahrnoval studium dokumentace, na jehož základě byly navrženy testovací případy. Testovací případy byly následně implementovány. Na závěr byla spuštěna jejich exekuce.

V rámci exekuce bylo provedeno 101 testů, z nichž 93 bylo vyhodnoceno jako passed a 8 jako failed. Exekuce testů proběhla automatizovaně s využitím nástroje Collection Runner. Procentuální vyjádření testů vyhodnocených jako passed je 92,08 %. Veškeré chyby byly nalezeny v rámci negativních testů a neměly by mít významný dopad na koncové uživatele. Možnou příčinou chyby může být špatně sestavený požadavek nebo chyba v kódu na straně serverové aplikace.

Diplomová práce si kladla za hlavní cíl vytvoření manuálu pro testování API s aplikací Postman. Manuál byl vytvořen jako součást diplomové práce v příloze (Příloha C). Poskytuje podrobné informace a návod pro úspěšné API testování s využitím nástroje Postman.

Manuál se věnuje přípravě a odesílání requestů a jejich organizaci v collections. Blíže popisuje možnosti exportování, importování a sdílení collections mezi více uživatelů. V rámci manuálu je vysvětleno, jaké typy proměnných Postman obsahuje a jak s nimi uživatel může pracovat, aby je mohl použít v testovacích skriptech. Manuál popisuje tvorbu testovacích skriptů, automatizaci testů s pomocí nástroje Collection Runner a možnosti monitoringu.

Cíle práce byl dosaženo, jelikož byl vytvořen ucelený manuál, který by měl být užitečný pro všechny, kteří se zajímají o API testování a uvažují o užití nástroje Postman.

ZÁVĚR

Diplomová práce byla zaměřena na integrační testování s využitím nástroje Postman. Hlavním cílem bylo vytvořit manuál pro automatizované testování aplikačních rozhraní s použitím testovacího softwaru Postman.

První kapitola se věnuje současnému stavu testování softwaru. Popisuje historii testování, jednotlivé statické a dynamické testovací techniky a rozdíly mezi nimi. Dále rozebírá, jaké postavení má testování v rámci různých modelů životního cyklu vývoje softwaru.

V závěru první kapitoly byly objasněny úrovně testování, a to testování jednotek, integrační testování, systémové testování a akceptační testování.

Třetí kapitola popisuje HTTP protokol, metody HTTP požadavků a stavové kódy. Dále se práce zaobírá aplikačním rozhraním, které umožňuje komunikaci mezi různými aplikacemi. V rámci této části byl také popsán testovací nástroj Postman, z hlediska toho, jaké funkce nabízí, jakými způsoby lze aplikaci používat a jaké další nástroje pro testování API v současnosti na trhu existují.

V praktické části bylo testování aplikačního rozhraní znázorněno na veřejně dostupném API. V rámci příkladu byly navrženy testovací případy, implementovány testovací skripty a následně spuštěna a vyhodnocena jejich exekuce. Závěrem byl vytvořen manuál k aplikaci Postman, který je uveden v příloze (Příloha C) diplomové práce.

Vytvořený dokument si klade za cíl seznámit uživatele s možnostmi využití tohoto nástroje pro testování API. Manuál obsahuje návody a ukázky, na jejichž základě by měl být uživatel schopný aplikaci Postman používat dle svých specifických potřeb, aplikovat nabyté znalosti k automatizaci testů a povznést tak celkovou kvalitu testování a zvyšovat kvalitu vyvíjeného produktu.

POUŽITÁ LITERATURA

[1] GELPERIN, David a Bill HETZEL. *The growth of software testing: Communications of the ACM* [online]. 1988, 687-695 [cit. 2022-10-16]. Dostupné z: https://www.researchgate.net/publication/234808293_The_growth_of_software_testing

[2] ULLAH, Sami. A brief history of software testing. *Salsa Digital* [online]. Suite 24, 3 Albert Coates Lane, MELBOURNE VIC 3000, Australia, 3. 12. 2019 [cit. 2022-10-16]. Dostupné z: https://salsa.digital/insights/a-brief-history-of-software-testing

[3] Types of Static Testing, 2013. *GeeksforGeeks* [online]. A-143, 9th Floor, Sovereign Corporate Tower, Sector-136, Noida, Uttar Pradesh, 13. 7. 2020 [cit. 2022-10-16]. Dostupné z: https://www.geeksforgeeks.org/types-of-static-testing/

[4] SAURAV, Sanket, 2022. The exponential cost of fixing bugs. *DeepSource* [online]. 29. 1. 2019 [cit. 2022-10-16]. Dostupné z: https://deepsource.io/blog/exponential-cost-of-fixing-bugs/

[5] MÜLLER, Thomas, 2013. Učební osnovy – Certifikovaný tester základní úrovně (verze 2018 v3.1): International Software Testing Qualifications Board. *Czech and Slovak Testing Board* [online]. 10. 3. 2020 [cit. 2022-10-16]. Dostupné z: https://castb.org/wp-content/uploads/2020/05/ISTQB_CTFL_CZ_3_1_1-6.pdf

[6] PUGH, W., N. AYEWAH, D. HOVEMEYER, J. D. MORGENTHALER a J. PENIX. *Using Static Analysis to Find Bugs* [online]. IEEE Software, 8. 10. 2008, 22-29 [cit. 2022-10-16]. Dostupné z: doi:10.1109/MS.2008.130

[7] BELLAIRS, Richard, 2022. What Is Static Code Analysis? Static Code Analysis Overview. *Perforce Software* [online]. 10. 2. 2020 [cit. 2022-10-16]. Dostupné z: https://www.perforce.com/blog/sca/what-static-analysis

[8] TOP 40 Static Code Analysis Tools (Best Source Code Analysis Tools). *SOFTWARE TESTING HELP* [online]. 24. 9. 2022 [cit. 2022-10-16]. Dostupné z: https://www.softwaretestinghelp.com/tools/top-40-static-code-analysis-tools/

[9] Software Testing Methodologies For Robust Software Delivery. *SOFTWARE TESTING HELP* [online]. 29. 9. 2022 [cit. 2022-10-16]. Dostupné z: https://www.softwaretestinghelp.com/software-development-testing-methodologies/

[10] SACHEDINA, Fahim, 2022. What is Exploratory Testing? | Global App Testing. *Global App Testing* [online]. England [cit. 2022-10-16]. Dostupné z: https://www.globalapptesting.com/blog/what-is-exploratory-testing

[11] KITNER, Ing. Radek, 2021. Přehled testovacích technik. *KITNER* [online]. Přízřenická 1023, 664 42, Modřice, Česká republika [cit. 2022-10-16]. Dostupné z: https://kitner.cz/testovani_softwaru/prehled-testovacich-technik/

[12] MARTINŮ, Mgr. Jiří a doc. Ing. Petr ČERMÁK, PH.D., 2018. *Metodiky vývoje softwaru* [online]. Olomouc [cit. 2022-10-16]. Dostupné z: https://dl1.cuni.cz/pluginfile.php/864918/mod_resource/content/1/Metodiky-v%C3%BDvoje-software-studijn%C3%AD-text.pdf. Studijní podpora pro kombinované studium. Moravská vysoká škola Olomouc.

[13] PATTON, Ron, 2002. Testování softwaru. Praha: Computer Press. Programování. ISBN 80-7226-636-5.

[14] BOEHM, Barry W. A Spiral Model of Software Development and Enhancement. TRW Defense Syst. Group, Redondo Beach, CA, roč. 21, č. 5, August 1988. ISSN: 0018-9162

[15] HLAVA, Tomáš. Spirálový model. *Testování softwaru: Portál zabývající se problematikou ověřování kvality software. Manuální i automatizované testování.* [online]. [cit. 2022-10-16]. Dostupné z: http://testovanisoftwaru.cz/manualni-testovani/modely-zivotniho-cyklu-softwaru/spiralovy-model/

[16] DEFINITION: Spiral model, 2022. *TechTarget* [online]. srpen 2019 [cit. 2022-10-16]. Dostupné z: https://www.techtarget.com/searchsoftwarequality/definition/spiral-model

[17] Difference between V-model and Waterfall model, 2013. *GeeksforGeeks* [online]. A-143, 9th Floor, Sovereign Corporate Tower, Sector-136, Noida, Uttar Pradesh, 23. 8. 2022 [cit. 2022-10-16]. Dostupné z: https://www.geeksforgeeks.org/difference-between-v-model-and-waterfall-model/

[18] SDLC – V-Model, 2022. *Tutorialspoint: simply easy learning* [online]. India Private Limited, 4th Floor, Incor9 Building, Plot No: 283/A, Kavuri Hills, Madhapur, Hyderabad, Telangana, INDIA-500081 [cit. 2022-10-16]. Dostupné z: https://www.tutorialspoint.com/sdlc/sdlc_v_model.htm

[19] Software Engineering | W-Model, 2013. *GeeksforGeeks* [online]. A-143, 9th Floor, Sovereign Corporate Tower, Sector-136, Noida, Uttar Pradesh, 28. 6. 2020 [cit. 2022-10-16]. Dostupné z: https://www.geeksforgeeks.org/software-engineering-w-model/

[20] JANSE, B., 2022. Rational Unified Process (RUP). *Toolshero* [online]. 07. 03. 2022 [cit. 2022-10-16]. Dostupné z: https://www.toolshero.com/information-technology/rational-unified-process-rup/

[21] KRUCHTEN, Philippe, 2004. The rational unified process: an introduction. 3rd ed. Upper Saddle River: Addison-Wesley. ISBN 0-321-19770-4.

[22] SCHWABER, Ken a Jeff SUTHERLAND, 2020. The Scrum Guide: The Definitive Guide to Scrum: The Rules of the Game. *ScrumGuides* [online]. listopad 2020 [cit. 2022-10-16]. Dostupné z: https://scrumguides.org/docs/scrumguide/v2020/2020-Scrum-Guide-US.pdf

[23] What is Scrum? A Better Way Of Building Products, 2022. *Scrum* [online]. [cit. 2022-10-16]. Dostupné z: https://www.scrum.org/resources/what-is-scrum

[24] Scrum Testing: A Detailed Guide to Testing on an Agile Team, 2022. *Testim* [online]. 04. 09. 2019 [cit. 2022-10-16]. Dostupné z: https://www.testim.io/blog/scrum-testing-guide/

[25] RAD (Rapid Application Development) Model. *Javatpoint* [online]. [cit. 2022-10-16]. Dostupné z: https://www.javatpoint.com/software-engineering-rapid-application-development-model

[26] SDLC RAD model. *W3schools® of Technology* [online]. 2022 [cit. 2022-10-16]. Dostupné z: https://www.w3schools.in/sdlc/rad-model

[27] Software Engineering | Rapid application development model (RAD), 2013. *GeeksforGeeks* [online]. A-143, 9th Floor, Sovereign Corporate Tower, Sector-136, Noida, Uttar Pradesh, 30. 05. 2022 [cit. 2022-10-16]. Dostupné z: https://www.geeksforgeeks.org/software-engineering-rapid-application-development-model-rad/

[28] BLACK, Rex, 2009. *Managing the Testing Process: Practical Tools and Techniques for Managing Hardware and Software Testing*. 3rd Edition. 10475 Crosspoint Boulevard Indianapolis, IN 46256: Wiley Publishing. ISBN 978-0470404157.

[29] STEPHENS, Matt a Doug ROSENBERG, 2011. Testování softwaru řízené návrhem. Brno: Computer Press. ISBN 978-80-251-3607-2.

[30] Unit Testing. *Software Testing Fundamentals* [online]. 29. 08. 2022 [cit. 2022-10-16]. Dostupné z: https://softwaretestingfundamentals.com/unit-testing/

[31] TERRA, John. What is Integration Testing: Examples, Challenges, and Approaches. *Simplilearn* [online]. 02. 08. 2022 [cit. 2022-10-16]. Dostupné z: https://www.simplilearn.com/what-is-integration-testing-examples-challenges-approaches-article

[32] HAMILTON, Thomas, 2022. Integration Testing: What is, Types with Example. *Guru99* [online]. 27. 08. 2022 [cit. 2022-10-16]. Dostupné z: https://www.guru99.com/integration-testing.html

[33] KRALJ, Kristijan, 2023. Integration Testing vs API Testing: Know the Differences. *MethodPoet* [online]. [cit. 2023-03-26]. Dostupné z: https://methodpoet.com/integration-testing-vs-api-testing/

[34] BUREŠ, Miroslav, Miroslav RENDA, Michal DOLEŽEL, Peter SVOBODA, Zdeněk GRÖSSL, Martin KOMÁREK, Ondřej MACEK a Radoslav MLYNÁŘ, 2016. Efektivní testování softwaru: klíčové otázky pro efektivitu testovacího procesu. Praha: Grada. Profesionál. ISBN 978-80-247-5594-6.

[35] SELVAN, Arul. An Introduction to Integration Testing and its Types: 23. 05. 2020. *TechAffinity* [online]. 20701 Bruce B Downs Blvd, Suite 204 Tampa, FL 33647: TechAffinity [cit. 2022-10-16]. Dostupné z: https://techaffinity.com/blog/what-is-integration-testing/

[36] PEZZÈ, Mauro a Michal YOUNG, 2008. *Software Testing and Analysis: Process, Principles, and Techniques*. United States of America: John Wiley & Sons. ISBN 978-0-471-45593-6.

[37] What Is System Testing: A Ultimate Beginner's Guide, 2022. *Software Testing Fundamentals* [online]. 12. 02. 2023 [cit. 2023-03.-26]. Dostupné z: https://www.softwaretestinghelp.com/system-testing/

[38] RANKL, Wolfgang a Wolfgang EFFING, 2010. Smart card handbook. 4th ed. Chichester: John Wiley. ISBN 978-0-470-74367-6.

[39] HTTP request methods. MDN [online]. 9. 9. 2022 [cit. 2023-01-21]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods

[40] HTTP response status codes. *MDN* [online]. 17. 1. 2023 [cit. 2023-01-21]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/HTTP/Status

[41] RICHARDSON, Alan, 2017. Automating and Testing a REST API: A Case Study in API testing using: Java, REST Assured, Postman, Tracks, cURL and HTTP Proxies. Compendium Developments. ISBN 978-0956733290.

[42] RICHARDSON, Leonard, Michael AMUNDSEN a Sam RUBY, 2013. *RESTful Web APIs*. Sebastopol: O'Reilly. ISBN 978-1-4493-5806-8.

[43] SNELL, James, Doug TIDWELL a Pavel KULCHENKO, 2002. Programming Web services with SOAP. Beijing: O'Reilly. ISBN 0-596-00095-2.

[44] SURENDRAKUMAR WARVANTE, Madhuri. A Survey on Developing Web Services with SOAP and REST. *International Journal of Advanced Research in Computer and Communication Engineering* [online]. October 2017, 7 [cit. 2023-02-24]. ISSN 2278-1021. Dostupné z: doi:10.17148/IJARCCE.2017.61020

[45] KOSEK, Jiří, 2002. *Inteligentní podpora navigace na WWW s využitím XML*. Praha. Diplomová práce. VYSOKÁ ŠKOLA EKONOMICKÁ V PRAZE. Vedoucí práce Ing. Vojtěch Svátek, Dr.

[46] PORCELLO, Eve a Alex BANKS, 2018. *Learning GraphQL: declarative data fletching for modern web apps*. Sebastopol,CA: O´Reilly. ISBN 978-1-492-03071-3.

[47] WESTERVELD, Dave, 2021. API Testing and Development with Postman: A practical guide to creating, testing, and managing APIs for automated software testing. Birmingham, England: Packt Publishing. ISBN 978-1800569201.

[48] INDRASIRI, Kasun, 2020. *GRPC: Up and Running: Building Cloud Native Applications with Go and Java for Docker and Kubernetes*. O'Reilly Media. ISBN 978-1492058335.

[49] JOHNSTON-GILBERT, Rebecca, 2023. There's an API for that. Postman: Postman, Inc. [online]. San Francisco: Postman, 13. 2. 2023 [cit. 2023-03-26]. Dostupné z: https://blog.postman.com/theres-an-api-for-that/

[50] Navigating Postman, 2023. *Postman: Postman, Inc.* [online]. San Francisco: Postman, 22. 2. 2023 [cit. 2023-02-24]. Dostupné z: https://learning.postman.com/docs/getting-started/navigating-postman/

[51] Postman API Platform Plans and Pricing: Plan comparison, 2023. *Postman: Postman, Inc.* [online]. San Francisco: Postman [cit. 2023-02-24]. Dostupné z: https://www.postman.com/pricing/

[52] Introduction to Insomnia, 2021. *Insomnia Docs by Kong* [online]. Kong [cit. 2023-03-26]. Dostupné z: https://docs.insomnia.rest/insomnia/get-started

[53] HARIKA, Asapu, 2023. SoapUI vs Postman: Difference Between SoapUI vs Postman. *EDUCBA* [online]. [cit. 2023-03-26]. Dostupné z: https://www.educba.com/soapui-vs-postman/

[54] Apache JMeter™, 2022. *The Apache Software Foundation* [online]. Apache Software Foundation [cit. 2023-03-26]. Dostupné z: https://jmeter.apache.org/

[55] Popis veřejných webových služeb. *Česká pošta* [online]. [cit. 2023-05-13]. Dostupné z: https://b2c.cpost.cz/

[56] Postman. Postman for Windows. Version 10.9.4 [software]. [cit. 2023-04-11]. Dostupné z: https://www.postman.com/downloads/. Požadavky na systém: Windows 64-bit; velikost 165 MB.

PŘÍLOHY

Příloha A – Navržené testovací případy

Příloha B – Implementované testovací případy

Příloha C – Návod k aplikaci Postman

Příloha A – Navržené testovací případy

2) Návrh testovacích případů pro getDistrictList

POZITIVNÍ TESTY:

a) Test, který ověří, že vrácená odpověď obsahuje pole, které není prázdné.

b) Test, který ověří, že každý objekt v poli obsahuje pouze vlastnosti id a name.

c) Test, který ověří, že hodnoty vlastností id jsou datového typu string.

d) Test, který ověří, že hodnota vlastnosti name v každém objektu obsahuje očekávané znaky, přičemž první písmeno je velké.

e) Test, který ověří, že hodnota stavového kódu při validním vstupu je 200.

f) Test, který ověří, že API nevrací duplicitní hodnoty.

g) Test, který ověří, že Address/getDistrictListAsXml vrací stejné hodnoty jako Address/getDistrictListAsJson.

NEGATIVNÍ TESTY:

h) Test, který ověří, že při použití HTTP metody POST bude vrácen stavový kód 405.

i) Test, který ověří, že stavový kód je 400 při odeslání requestu bez povinného parametru id.

j) Test, který ověří, že stavový kód je 400 při odeslání requestu s nevalidní hodnotou parametru id.

k) Test, který ověří, že stavový kód je 400 při odeslání requestu s prázdnou hodnotou parametru id.

3) Návrh testovacích případů pro getCityList

POZITIVNÍ TESTY:

a) Test, který ověří, že vrácená odpověď obsahuje pole, které není prázdné.

b) Test, který ověří, že každý objekt v poli obsahuje pouze vlastnosti id a name.

c) Test, který ověří, že hodnoty vlastností id jsou datového typu string.

d) Test, který ověří, že hodnota vlastnosti name v každém objektu obsahuje očekávané znaky, přičemž první písmeno je velké.

e) Test, který ověří, že hodnota stavového kódu při validním vstupu je 200.

f) Test, který ověří, že API nevrací duplicitní hodnoty.

g) Test, který ověří, že Address/getCityListAsXml vrací stejné hodnoty jako Address/getCityListAsJson.

NEGATIVNÍ TESTY:

h) Test, který ověří, že při použití HTTP metody POST bude vrácen stavový kód 405.

i) Test, který ověří, že stavový kód je 400 při odeslání requestu bez povinného parametru id.

j) Test, který ověří, že stavový kód je 400 při odeslání requestu s nevalidní hodnotou parametru id.

k) Test, který ověří, že stavový kód je 400 při odeslání requestu s prázdnou hodnotou parametru id.

4) Návrh testovacích případů pro getCityPartList

POZITIVNÍ TESTY:

a) Test, který ověří, že každý objekt v poli obsahuje pouze vlastnosti id a name.

b) Test, který ověří, že hodnoty vlastností id jsou datového typu string.

c) Test, který ověří, že hodnota stavového kódu při validním vstupu je 200.

d) Test, který ověří, že API nevrací duplicitní hodnoty.

e) Test, který ověří, že Address/getCityPartListAsXml vrací stejné hodnoty jako Address/getCityPartListAsJson.

NEGATIVNÍ TESTY:

f) Test, který ověří, že při použití HTTP metody POST bude vrácen stavový kód 405.

g) Test, který ověří, že stavový kód je 400 při odeslání requestu bez povinného parametru id.

h) Test, který ověří, že stavový kód je 400 při odeslání requestu s nevalidní hodnotou parametru id.

i) Test, který ověří, že stavový kód je 400 při odeslání requestu s prázdnou hodnotou parametru id.

5) Návrh testovacích případů pro getStreetList

POZITIVNÍ TESTY:

a) Test, který ověří, že každý objekt v poli obsahuje pouze vlastnosti id a name.

b) Test, který ověří, že hodnoty vlastností id a name jsou datového typu string.

c) Test, který ověří, že hodnota stavového kódu při validním vstupu je 200.

d) Test, který ověří, že API nevrací duplicitní hodnoty.

e) Test, který ověří, že Address/getStreetListAsXml vrací stejné hodnoty jako Address/getStreetListAsJson.

NEGATIVNÍ TESTY:

f) Test, který ověří, že při použití HTTP metody POST bude vrácen stavový kód 405.

g) Test, který ověří, že stavový kód je 400 při odeslání requestu bez povinného parametru id.

h) Test, který ověří, že stavový kód je 400 při odeslání requestu s nevalidní hodnotou parametru id.

i) Test, který ověří, že stavový kód je 400 při odeslání requestu s prázdnou hodnotou parametru id.

6) Návrh testovacích případů pro getNumberList

POZITIVNÍ TESTY:

a) Test, který ověří, že při použití query parametru idCityPart každý objekt v poli obsahuje pouze vlastnosti id a name.

b) Test, který ověří, že při použití query parametru idStreet každý objekt v poli obsahuje pouze vlastnosti id a name.

c) Test, který ověří, že při použití query parametru idCityPart jsou hodnoty vlastností id datového typu string.

d) Test, který ověří, že při použití query parametru idStreet jsou hodnoty vlastností id datového typu string.

e) Test, který ověří, že hodnoty vlastností id jsou při použití query parametru idCityPart datového typu string.

f) Test, který ověří, že hodnoty vlastností id jsou při použití query parametru idStreet datového typu string.

g) Test, který ověří, že hodnota stavového kódu při validním vstupu při použití query parametru idCityPart je 200.

h) Test, který ověří, že hodnota stavového kódu při validním vstupu při použití query parametru idStreet je 200.

i) Test, který ověří, že API nevrací duplicitní hodnoty při použití query parametru idCityPart.

j) Test, který ověří, že API nevrací duplicitní hodnoty při použití query parametru idStreet.

k) Test, který ověří, že Address/getNumberListAsXml vrací stejné hodnoty jako Address/ getNumberListAsJson při použití query parametru idCityPart.

l) Test, který ověří, že Address/getNumberListAsXml vrací stejné hodnoty jako Address/ getNumberListAsJson při použití query parametru idStreet.

m) Test, který ověří, že v případě, pokud jsou uvedeny oba parametry idStreet a idCityPart, je upřednostněn parametr idStreet,

NEGATIVNÍ TESTY:

n) Test, který ověří, že při použití HTTP metody POST bude vrácen stavový kód 405.

o) Test, který ověří, že stavový kód je 400 při odeslání requestu bez povinného parametru idStreet.

p) Test, který ověří, že stavový kód je 400 při odeslání requestu bez povinného parametru idCityPart.

q) Test, který ověří, že stavový kód je 400 při odeslání requestu s nevalidní hodnotou parametru idCityPart.

r) Test, který ověří, že stavový kód je 400 při odeslání requestu s nevalidní hodnotou parametru idStreet.

s) Test, který ověří, že stavový kód je 400 při odeslání requestu s prázdnými hodnotami parametrů idStreet a idCityPart.

Příloha B – Implementované testovací případy

2) Implementované testovací skripty pro getDistrictList

a) Test, který ověří, že vrácená odpověď obsahuje pole, které není prázdné.

```
pm.test("Odpověď obsahuje pole, které není prázdné", function() {
    pm.expect(jsonData).to.be.an("array").that.is.not.empty;
});
```

b) Test, který ověří, že každý objekt v poli obsahuje pouze vlastnosti id a name.

```
pm.test("Každý objekt v poli obsahuje pouze vlastnosti id a name", function() {
    jsonData.forEach(function(district) {
        pm.expect(Object.keys(district)).to.have.members(["id", "name"]);
    });
});
```

c) Test, který ověří, že hodnoty vlastností id jsou datového typu string.

```
pm.test("Hodnoty vlastností id jsou datového typu string", function() {
    jsonData.forEach(function(district) {
        pm.expect(district.id).to.be.a("string").that.is.not.empty;
    });
});
```

d) Test, který ověří, že hodnota vlastnosti name v každém objektu obsahuje očekávané znaky, přičemž první písmeno je velké.

```
pm.test("Hodnota vlastnosti name v každém objektu obsahuje pouze písmena a první pí
smeno je vždy velké", function() {
    jsonData.forEach(function(district) {
        pm.expect(district.name).to.match(/^[A-ZŠČŘŽÝÁÍÉÚŮĚÔŇŤ][\p{L}\s-]*$/u);
    });
});
```

e) Test, který ověří, že hodnota stavového kódu při validním vstupu je 200.

```
pm.test("Stavový kód má hodnotu 200", function () {
    pm.response.to.have.status(200);
});
```

f) Test, který ověří, že API nevrací duplicitní hodnoty.

```
pm.test("API nevrací duplicitní hodnoty", function() {
    let ids = jsonData.map(function(idMapper) {
        return idMapper.id;
    });
    pm.expect(ids.length).to.equal(new Set(ids).size);
});
```

g) Test, který ověří, že Address/getDistrictListAsXml vrací stejné hodnoty jako Address/getDistrictListAsJson.

```
let responseJson = xml2Json(responseBody);

let districts = [];

if (Array.isArray(responseJson.districts.district)) {

    districts = responseJson.districts.district.map(district => {
        return {
            id: district.id,
            name: district.name
        };
    });
} else {
    districts = [{
        id: responseJson.districts.district.id,
        name: responseJson.districts.district.name
    }];
}
let classicJson = JSON.parse(pm.collectionVariables.get("districtJson"));

pm.test("XML odpověď vrací stejné hodnoty jako JSON odpověď.", function() {
    pm.expect(districts).to.eql(classicJson);
});
```

h) Test, který ověří, že při použití HTTP metody POST bude vrácen stavový kód 405.

```
pm.test("Stavový kód má hodnotu 405", function() {

    pm.expect(pm.response.code).to.eql(405);
});
```

i, j, k) Test, který ověří, že stavový kód je 400 při odeslání requestu bez povinného parametru id, při odeslání requestu s nevalidní hodnotou parametru id, při odeslání requestu s prázdnou hodnotou parametru id.

```
pm.test("Stavový kód má hodnotu 400", function() {
    pm.expect(pm.response.code).to.eql(400);
});
```

3) Implementované testovací skripty pro getCityList

a) Test, který ověří, že vrácená odpověď obsahuje pole, které není prázdné.

```
pm.test("Odpověď obsahuje pole, které není prázdné", function() {
    pm.expect(jsonData).to.be.an("array").that.is.not.empty;
});
```

b) Test, který ověří, že každý objekt v poli obsahuje pouze vlastnosti id a name.

```
pm.test("Každý objekt v poli obsahuje pouze vlastnosti id a name", function() {
    jsonData.forEach(function(city) {
        pm.expect(Object.keys(city)).to.have.members(["id", "name"]);
    });
});
```

c) Test, který ověří, že hodnoty vlastností id jsou datového typu string.

```
pm.test("Hodnoty vlastností id jsou datového typu string", function() {
    jsonData.forEach(function(city) {
        pm.expect(city.id).to.be.a("string").that.is.not.empty;
    });
});
```

d) Test, který ověří, že hodnota vlastnosti name v každém objektu obsahuje očekávané znaky, přičemž první písmeno je velké.

```
pm.test("Hodnota vlastnosti name v každém objektu obsahuje pouze písmena a první pí
smeno je vždy velké", function() {
    jsonData.forEach(function(city) {
        pm.expect(city.name).to.match(/^[A-ZŠČŘŽÝÁÍÉÚŮĚÔŇŤ][\p{L}\s-]*$/u);
    });
});
```

e) Test, který ověří, že hodnota stavového kódu při validním vstupu je 200.

```
pm.test("Stavový kód má hodnotu 200", function () {
    pm.response.to.have.status(200);
});
```

f) Test, který ověří, že API nevrací duplicitní hodnoty.

```
pm.test("API nevrací duplicitní hodnoty", function() {
    let ids = jsonData.map(function(idMapper) {
        return idMapper.id;
    });
    pm.expect(ids.length).to.equal(new Set(ids).size);
});
```

g) Test, který ověří, že Address/getCityListAsXml vrací stejné hodnoty jako Address/getCityListAsJson.

```
let responseJson = xml2Json(responseBody);

let cities = [];

if (Array.isArray(responseJson.cities.city)) {
    cities = responseJson.cities.city.map(city => {
        return {
            id: city.id,
            name: city.name
        };
    });
} else {
    cities = [{
        id: responseJson.cities.city.id,
        name: responseJson.cities.city.name
    }];
}

let classicJson = JSON.parse(pm.collectionVariables.get("cityJson"));

pm.test("XML odpověď vrací stejné hodnoty jako JSON odpověď.", function() {
    pm.expect(cities).to.eql(classicJson);
});
```

h) Test, který ověří, že při použití HTTP metody POST bude vrácen stavový kód 405.

```
pm.test("Stavový kód má hodnotu 405", function() {
    pm.expect(pm.response.code).to.eql(405);
});
```

i, j, k) Test, který ověří, že stavový kód je 400 při odeslání requestu bez povinného parametru id, při odeslání requestu s nevalidní hodnotou parametru id, při odeslání requestu s prázdnou hodnotou parametru id.

```
pm.test("Stavový kód má hodnotu 400", function() {
    pm.expect(pm.response.code).to.eql(400);
});
```

4) Implementované testovací skripty pro getCityPartList

a) Test, který ověří, že každý objekt v poli obsahuje pouze vlastnosti id a name.

```
pm.test("Každý objekt v poli obsahuje pouze vlastnosti id a name", function() {
    jsonData.forEach(function(cityPart) {
        pm.expect(Object.keys(cityPart)).to.have.members(["id", "name"]);
    });
});
```

b) Test, který ověří, že hodnoty vlastností id jsou datového typu string.

```
pm.test("Hodnoty vlastností id jsou datového typu string", function() {
    jsonData.forEach(function(cityPart) {
        pm.expect(cityPart.id).to.be.a("string").that.is.not.empty;
    });
});
```

c) Test, který ověří, že hodnota stavového kódu při validním vstupu je 200.

```
pm.test("Stavový kód má hodnotu 200", function () {
    pm.response.to.have.status(200);
});
```

d) Test, který ověří, že API nevrací duplicitní hodnoty.

```
pm.test("API nevrací duplicitní hodnoty", function() {
    let ids = jsonData.map(function(idMapper) {
        return idMapper.id;
    });
    pm.expect(ids.length).to.equal(new Set(ids).size);
});
```

e) Test, který ověří, že Address/getCityPartListAsXml vrací stejné hodnoty jako Address/getCityPartListAsJson.

```
let responseJson = xml2Json(responseBody);

let cityParts = [];


if (responseJson.cityParts && responseJson.cityParts.cityPart) {
  if (Array.isArray(responseJson.cityParts.cityPart)) {
    cityParts = responseJson.cityParts.cityPart.map(cityPart => {
      return {
        id: cityPart.id,
        name: cityPart.name
      };
    });
  } else {
    cityParts = [{
      id: responseJson.cityParts.cityPart.id,
      name: responseJson.cityParts.cityPart.name
    }];
  }
}

let classicJson = JSON.parse(pm.collectionVariables.get("cityPartJson"));

pm.test("XML odpověď vrací stejné hodnoty jako JSON odpověď.", function() {
    pm.expect(cityParts).to.eql(classicJson);
});
```

f) Test, který ověří, že při použití HTTP metody POST bude vrácen stavový kód 405.

```
pm.test("Stavový kód má hodnotu 405", function() {
    pm.expect(pm.response.code).to.eql(405);
});
```

g, h, i) Test, který ověří, že stavový kód je 400 při odeslání requestu bez povinného parametru id, při odeslání requestu s nevalidní hodnotou parametru id, při odeslání requestu s prázdnou hodnotou parametru id.

```
pm.test("Stavový kód má hodnotu 400", function() {
    pm.expect(pm.response.code).to.eql(400);
});
```

5) Implementované testovací skripty pro getStreetList

a, b) Test, který ověří, že každý objekt v poli obsahuje pouze vlastnosti id a name, hodnoty vlastností id jsou datového typu string.

```
let jsonData = pm.response.json();

if (jsonData !== undefined && jsonData !== null && jsonData.length > 0 ) {
    let streetId = jsonData[0].id;
    pm.collectionVariables.set("streetId", streetId);

    pm.test("Každý objekt v poli obsahuje vlastnost name", function() {
        jsonData.forEach(function(street) {
            pm.expect(street).to.have.property("name").that.is.not.empty;
        });
    });

    pm.test("Každý objekt v poli obsahuje vlastnost id", function() {
        jsonData.forEach(function(street) {
            pm.expect(street).to.have.property("id").that.is.not.empty;
        });
    });

    pm.test("Hodnoty vlastností id jsou datového typu string", function() {
        jsonData.forEach(function(street) {
            pm.expect(street.id).to.be.a("string").that.is.not.empty;
        });
    });

    pm.test("Hodnoty vlastností name jsou datového typu string", function() {
        jsonData.forEach(function(street) {
            pm.expect(street.name).to.be.a("string").that.is.not.empty;
        });
    });
}
```

c) Test, který ověří, že hodnota stavového kódu při validním vstupu je 200.

```
pm.test("Stavový kód má hodnotu 200", function () {
    pm.response.to.have.status(200);
});
```

d) Test, který ověří, že API nevrací duplicitní hodnoty.

```
pm.test("API nevrací duplicitní hodnoty", function() {

    let ids = jsonData.map(function(idMapper) {
        return idMapper.id;
    });
    pm.expect(ids.length).to.equal(new Set(ids).size);
});
```

e) Test, který ověří, že Address/getStreetListAsXml vrací stejné hodnoty jako Address/getStreetListAsJson

```
let responseJson = xml2Json(responseBody);

let streets = [];

if (responseJson.streets && responseJson.streets.street) {
  if (Array.isArray(responseJson.streets.street)) {
    streets = responseJson.streets.street.map(street => {
      return {
        id: street.id,
        name: street.name
      };
    });
  } else {
    streets = [{
      id: responseJson.streets.street.id,
      name: responseJson.streets.street.name
    }];
  }
}

let classicJson = JSON.parse(pm.collectionVariables.get("streetJson"));

pm.test("XML odpověď vrací stejné hodnoty jako JSON odpověď.", function() {
    pm.expect(streets).to.eql(classicJson);
});
```

f) Test, který ověří, že při použití HTTP metody POST bude vrácen stavový kód 405.

```
pm.test("Stavový kód má hodnotu 405", function() {
    pm.expect(pm.response.code).to.eql(405);
});
```

g, h, i) Test, který ověří, že stavový kód je 400 při odeslání requestu bez povinného parametru id, při odeslání requestu s nevalidní hodnotou parametru id, při odeslání requestu s prázdnou hodnotou parametru id.

```
pm.test("Stavový kód má hodnotu 400", function() {
    pm.expect(pm.response.code).to.eql(400);
});
```

6) Implementované testovací skripty pro getNumberList

a, b) Test, který ověří, že při použití query parametru idCityPart nebo idStreet každý objekt v poli obsahuje pouze vlastnosti id a name.

```
pm.test("Každý objekt v poli obsahuje pouze vlastnosti id a name", function() {
    jsonData.forEach(function(number) {
        pm.expect(Object.keys(number)).to.have.members(["id", "name"]);
    });
});
```

c, d) Test, který ověří, že při použití query parametru idCityPart nebo idStreet jsou hodnoty vlastností id datového typu string.

```
pm.test("Hodnoty vlastností id jsou datového typu string", function() {
    jsonData.forEach(function(number) {
        pm.expect(number.id).to.be.a("string").that.is.not.empty;
    });
});
```

e, f) Test, který ověří, že hodnoty vlastností id jsou při použití query parametru idCityPart nebo idStreet datového typu string.

```
pm.test("Hodnoty vlastností name jsou datového typu string", function() {
    jsonData.forEach(function(number) {
        pm.expect(number.name).to.be.a("string").that.is.not.empty;
    });
});
```

g, h) Test, který ověří, že hodnota stavového kódu při validním vstupu při použití query parametru idCityPart je 200. Test, který ověří, že hodnota stavového kódu při validním vstupu při použití query parametru idStreet je 200.

```
pm.test("Stavový kód má hodnotu 200", function () {
    pm.response.to.have.status(200);
});
```

i, j) Test, který ověří, že API nevrací duplicitní hodnoty při použití query parametru idCityPart. Test, který ověří, že API nevrací duplicitní hodnoty při použití query parametru idStreet.

```
pm.test("API nevrací duplicitní hodnoty", function() {

    let ids = jsonData.map(function(idMapper) {
        return idMapper.id;
    });
    pm.expect(ids.length).to.equal(new Set(ids).size);
});
```

k, l) Test, který ověří, že Address/getNumberListAsXml vrací stejné hodnoty jako Address/ getNumberListAsJson při použití query parametru idCityPart. Test, který ověří, že Address/getNumberListAsXml vrací stejné hodnoty jako Address/ getNumberListAsJson při použití query parametru idStreet.

```
let responseJson = xml2Json(responseBody);

let addresses = [];

if (Array.isArray(responseJson.addresses.address)) {
    addresses = responseJson.addresses.address.map(address => {
        return {
            id: address.id,
            name: address.name
        };
    });
} else {
    addresses = [{
        id: responseJson.addresses.address.id,
        name: responseJson.addresses.address.name
    }];
}

let classicJson = JSON.parse(pm.collectionVariables.get("numberJson"));
pm.test("XML odpověď vrací stejné hodnoty jako JSON odpověď.", function () {
    pm.expect(addresses).to.eql(classicJson);
});
```

m) Test, který ověří, že v případě, pokud jsou uvedeny oba parametry idStreet a idCityPart, je upřednostněn parametr idStreet,

```
pm.test("Pokud jsou oba parametry uvedeny, je upřednostněn parametr idStreet", func
tion () {
    pm.expect(pm.collectionVariables.get("numberListJson_1")).to.eql(pm.collectionV
ariables.get("numberListJson_2"));
});
```

n) Test, který ověří, že při použití HTTP metody POST bude vrácen stavový kód 405.

```
pm.test("Stavový kód má hodnotu 405", function() {
    pm.expect(pm.response.code).to.eql(405);
});
```

o, p, q, r, s) Test, který ověří, že stavový kód je 400 při odeslání requestu bez povinného parametru idStreet. Test, který ověří, že stavový kód je 400 při odeslání requestu bez povinného parametru idCityPart. Test, který ověří, že stavový kód je 400 při odeslání requestu s nevalidní hodnotou parametru idCityPart. Test, který ověří, že stavový kód je 400 při odeslání requestu s nevalidní hodnotou parametru idStreet. Test, který ověří, že stavový kód je 400 při odeslání requestu s prázdnými hodnotami parametrů idStreet a idCityPart.

```
pm.test("Stavový kód má hodnotu 400", function() {
    pm.expect(pm.response.code).to.eql(400);
});
```

Univerzita Pardubice, Fakulta ekonomicko-správní

Návod k aplikaci Postman

Praktická část diplomové práce

Stanislav Huňáček

2023

OBSAH

SEZNAM OBRÁZKŮ **3**

ÚVOD **4**

SEZNAM POUŽITÝCH IKON **5**

1 PŘEDPOKLÁDANÉ ZNALOSTI **6**

2 STAŽENÍ A INSTALACE APLIKACE **7**

3 PŘÍPRAVA A ODESLÁNÍ REQUESTU **9**

4 PRÁCE S COLLECTIONS **13**

5 EXPORTOVÁNÍ, IMPORTOVÁNÍ A SDÍLENÍ COLLECTIONS **16**

5.1 EXPORT COLLECTION 16

5.2 IMPORT COLLECTION 18

5.3 SDÍLENÍ COLLECTION 19

6 PRÁCE S PROMĚNNÝMI **21**

7 PŘÍPRAVA TESTOVACÍCH SKRIPTŮ PRO TESTOVÁNÍ API **25**

7.1 TESTY NA STAVOVÉ KÓDY 25

7.2 COLLECTION RUNNER 28

7.3 MONITORING API 31

7.4 TESTOVÁNÍ TĚLA ODPOVĚDI 34

8 DATA-DRIVEN TESTOVÁNÍ **37**

9 NEWMAN **40**

POUŽITÁ LITERATURA **43**

SEZNAM OBRÁZKŮ

Obrázek 1 – Stažení instalačního souboru....7
Obrázek 2 - Vytváření účtu....8
Obrázek 3 – Přechod do MyWorkspaces....9
Obrázek 4 – Tlačítko pro zobrazení nabídky pracovních bloků....9
Obrázek 5 – Otevření nového HTTP requestu....10
Obrázek 6 – Vložení URL do requestu....10
Obrázek 7 – Headers requestu....11
Obrázek 8 – Odeslání requestu....11
Obrázek 9 – Odpověď requestu ve formátu JSON....12
Obrázek 10 – Ukládání requestu....13
Obrázek 11 – Vytváření nové collection....13
Obrázek 12 – Přiřazení requestu do collection....14
Obrázek 13 – Zobrazení uloženého requestu v collection....14
Obrázek 14 – Collection – View more actions....16
Obrázek 15 – Export collection do JSON souboru....17
Obrázek 16 – Struktura exportovaného JSON souboru....17
Obrázek 17 – Tlačítko Import....18
Obrázek 18 – Vkládání souboru k importu....18
Obrázek 19 – Import Collection do My Workspace....19
Obrázek 20 – Nabídka možností ke sdílení Collection....19
Obrázek 21 – Úrovně proměnných v aplikaci Postman....21
Obrázek 22 – Vytváření Collection proměnné....22
Obrázek 23 – Vytváření local variable....23
Obrázek 24 – Hodnota query parametru odeslaného requestu....23
Obrázek 25 – Vkládání testovacích skriptů v záložce Tests....25
Obrázek 26 – PASS – pozitivní výsledek testu....26
Obrázek 27 – Negativní výsledek testu....26
Obrázek 28 – Výsledek testu requestu s názvem 10 names....27
Obrázek 29 – Výsledek requestu 11 names....28
Obrázek 30 – Run collection....29
Obrázek 31 – Nastavení manuálního běhu Collection Runneru....30
Obrázek 32 – Výsledky testů Collection Runneru....31
Obrázek 33 – Plánování pravidelného běhu testů....32
Obrázek 34 – Výsledky naplánovaných testů....33
Obrázek 35 – E-mailová notifikace oznamující chybu....34
Obrázek 36 – Výsledek testu těla odpovědi....35
Obrázek 37 – Vstupní data pro data-driven testování....37
Obrázek 38 – Náhled testovacích dat....38
Obrázek 39 – Spuštění data-driven testů....38
Obrázek 40 – Výsledky data-driven testů....39
Obrázek 41 – Instalace Newman....40
Obrázek 42 – Generování access key ke collection....41
Obrázek 43 – Newman – výsledky testů....41
Obrázek 44 – Newman – popis selhání....42

ÚVOD

Tento dokument slouží jako návod, jak využít funkce aplikace Postman k testování API.

První kapitola seznamuje s předpokládanými znalostmi, kterými je vhodné disponovat, aby bylo možné plně využít potenciálu aplikace Postman. Kapitola **Stažení a instalace aplikace** objasňuje, jakými způsoby lze Postman používat a jak si aplikaci nainstalovat.

Další kapitola se věnuje tomu, jak si dle API dokumentace připravit a odeslat první request. Kapitola **Práce s collections** popisuje, jak vytvářet a jak do collection přidávat requesty a organizovat je. V další části je vysvětleno, jak exportovat a importovat vytvořené collections a sdílet je tak s jinými uživateli. Kapitola **Práce s proměnnými** na příkladech prezentuje, jaké jsou rozdíly mezi jednotlivými typy proměnných a jak s proměnnými pracovat.

Kapitola **Příprava testovacích skriptů pro testování API** obsahuje vysvětlení, kde a jak vytvářet testovací skripty, jak vytvořené testy automatizovat a hromadně spouštět s pomocí Collection Runneru nebo s pomocí monitoringu collections. Kapitola **Data-driven testování** uvádí, jak si lze připravit testovací data pro provádění data-driven testů. Poslední kapitola znázorňuje, jak spouštět běh collections s pomocí příkazů nástrojem Newman.

Text je doprovázen obrázky, které mají uživateli napomoct s orientací v aplikaci.

Návod byl tvořen pro verzi aplikace Postman byla 10.9.4. V průběhu času se může stát, že určité části dokumentu se již nebudou shodovat s nejnověji vydanou verzí. V případě, že taková situace nastane a nebudete si vědět rady, doporučuji navštívit oficiální dokumentaci aplikace Postman a prostudovat si danou oblast. Nicméně věřím, že minimálně principy zůstanou podobné a manuál pro vás i tak bude nadále užitečný.

Tento manuál byl vytvořen jako součást diplomové práce „Integrační testování s využitím nástroje Postman“.

SEZNAM POUŽITÝCH IKON

Ikona symbolizuje poznámku, která podrobněji popisuje konkrétní část textu.

Zdroj: [1]

Ikona symbolizuje úkol k dané kapitole.

Zdroj: [2]

Ikona symbolizuje tip na užitečné informace nebo rady, které vám mohou usnadnit práci.

Zdroj: [3]

Ikona symbolizuje úvodní část kapitoly.

Zdroj: [4]

1 PŘEDPOKLÁDANÉ ZNALOSTI

Minimálními předpokládanými znalostmi pro úspěšné zvládnutí tohoto návodu jsou základní znalosti práce s počítačem a s webovými aplikacemi.

Pro maximální využití funkcí aplikace Postman pro testování API je vhodné mít znalosti také v následujících oblastech:

- Znalost API, REST, SOAP.
- Znalost různých druhů formátů souborů – JSON, XML, HTML.
- Znalost HTTP protokolu – požadavek, odpověď, stavové kódy, hlavičky.
- Znalost HTTP metod GET, POST, PUT, DELETE.
- Znalost programovacího jazyka Javascript.
- Znalost základů testování softwaru.
- Zkušenosti s různými nástroji pro automatické spouštění testů – Jenkins, GitLab CI/CD, Azure Pipelines apod.
- Znalost anglického jazyka pro čtení dokumentace.

Vybraná doporučená videa poskytující základní informace:

Comparing web API types: SOAP, REST, GraphQL and RPC: https://youtu.be/NFw0HznpLlM

JSON Tutorial For Beginners – Full Course: https://youtu.be/IWcUJLUAO2A

HTTP Crash Course & Exploration: https://youtu.be/iYM2zFP3Zn0

Learn JavaScript – Full Course for Beginners: https://youtu.be/PkZNo7MFNFg

Software Testing Full Course: https://www.youtube.com/live/sO8eGL6SFsA

Další klíčovou dovedností je znalost funkcí aplikace Postman, a právě k tomu slouží tento návod.

2 STAŽENÍ A INSTALACE APLIKACE

Postman je k dispozici jako desktopová aplikace pro operační systémy Windows, Mac a Linux nebo jako webová aplikace s podporou pro prohlížeče Google Chrome, Mozilla Firefox, Microsoft Edge a Safari.

V tomto návodu budeme pracovat s desktopovou verzí aplikace Postman. Pokud plánujete využívat webovou verzi, je vhodné si k ní nainstalovat Postman desktop agent, který si lze stáhnout zde: https://www.postman.com/downloads/postman-agent/.

Instalace desktopové aplikace Postman krok za krokem:

1. Otevřete ve svém prohlížeči stránku https://www.postman.com/downloads/.
2. Dle svého operačního systému si zvolte správnou verzi instalačního souboru a následně si soubor stáhněte do svého počítače.

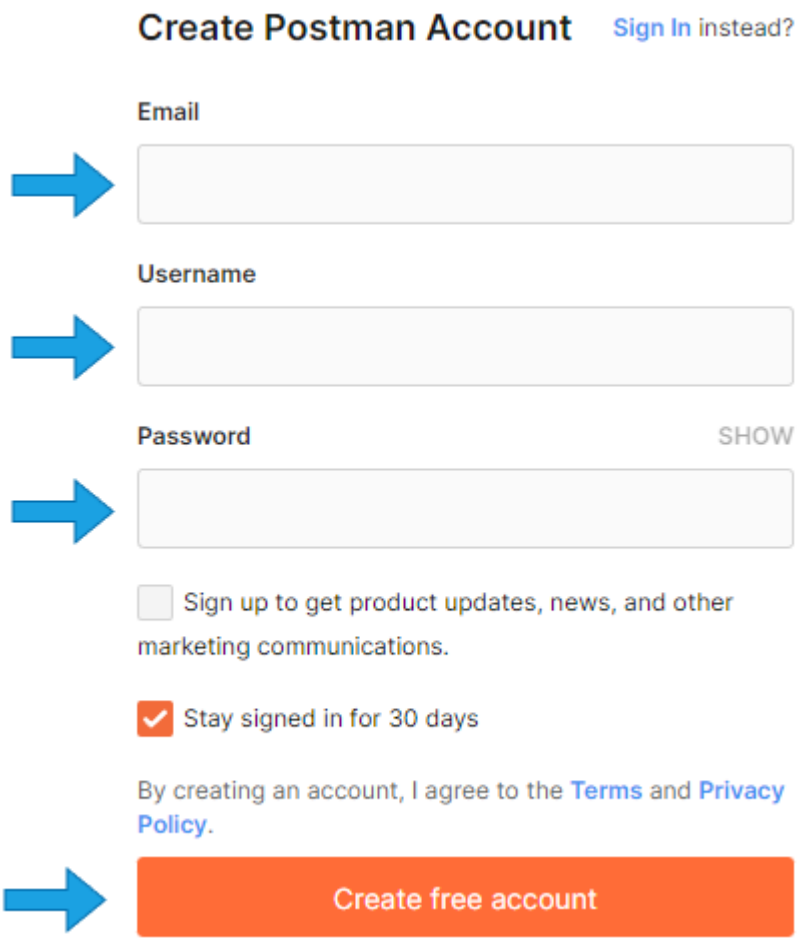
Obrázek 21 – Stažení instalačního souboru

Zdroj: [56]

3. Stažený soubor spusťte a počkejte na dokončení instalace.
4. Otevřete aplikaci Postman.
5. Po otevření aplikace se otevře výzva k zadání přihlašovacích údajů nebo k vytvoření účtu. Postman lze omezeně používat i bez vlastního účtu, avšak doporučuji účet založit,

což vám umožní plné využití všech funkcí, zejména funkce synchronizace, přístup do Workspaces a zálohování práce. Pokud aplikace nenabízí odkaz na vytvoření účtu, lze si účet založit zde – https://identity.getpostman.com/signup.

6. Na dané stránce vyplňte požadované údaje a klikněte na tlačítko **Create free account**.



Obrázek 22 - Vytváření účtu

Zdroj: [1]

7. Na zadanou e-mailovou adresu vám přijde ověřovací e-mail, který je nutné potvrdit.
8. S vytvořeným účtem se následně přihlaste do desktopové aplikace Postman.

Tip:

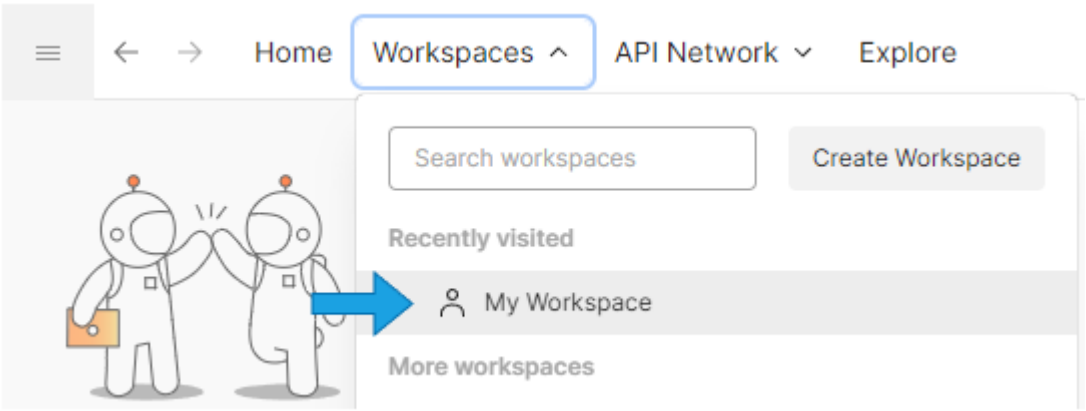
Po instalaci si nastavte barevné schéma aplikace dle svých preferencí. Toto nastavení najdete v **Settings** v záložce **Themes.**

3 PŘÍPRAVA A ODESLÁNÍ REQUESTU

V této kapitole si vyzkoušíte, jak vytvořit a odeslat svůj první request. Pro následující příklad bude využito API **Astronomy Picture of the Day**. Dokumentaci API lze najít na stránce https://go-apod.herokuapp.com/. Než se pustíte do dalších kroků, doporučuji si API dokumentaci prostudovat.

Po úspěšném přihlášení do aplikace si otevřete svůj workspace (pracovní prostor).

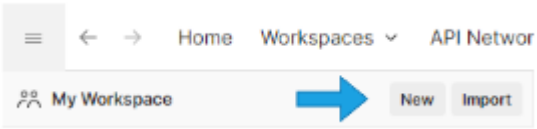
Klikněte na záložku **Workspaces** a vyberte **My Workspace**.



Obrázek 23 – Přechod do MyWorkspaces

Zdroj: [1]

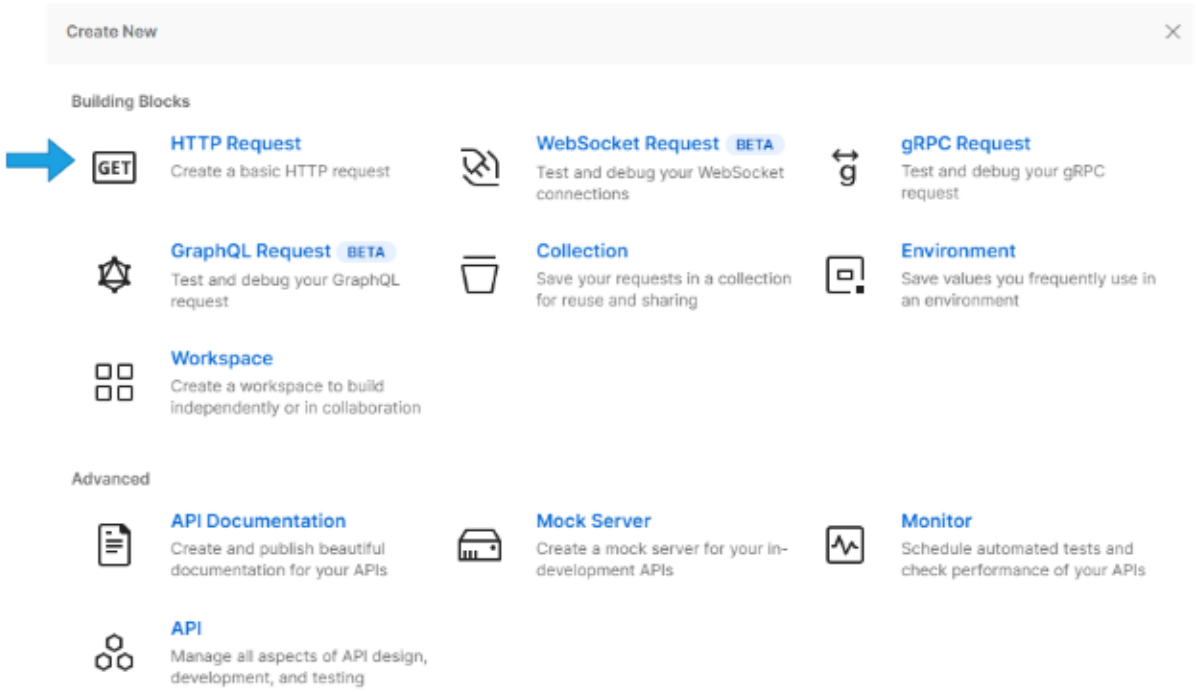
Nyní máte otevřené workspace **My Workspace**. Dále klikněte na tlačítko **New**.



Obrázek 24 – Tlačítko pro zobrazení nabídky pracovních bloků

Zdroj: [1]

Zobrazí se vám nabídka pracovních bloků. Z nabídky klikněte na **HTTP Request.**

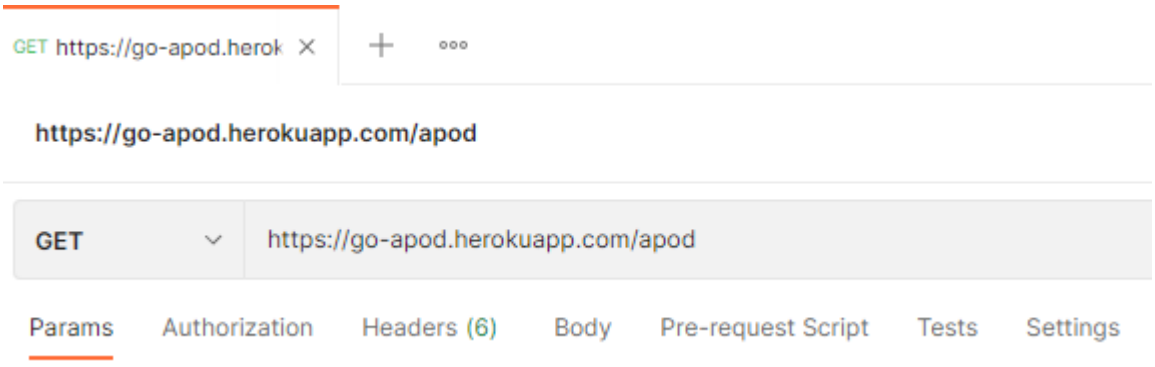


Obrázek 25 – Otevření nového HTTP requestu

Zdroj: [1]

Po kliknutí na **HTTP Request** se vám otevře nové okno s prázdným requestem.

Z API dokumentace lze vyčíst, že API specifikuje dva endpointy. Zkopírujte URL endpointu ze specifikace pro Full Info endpoint (https://go-apod.herokuapp.com/apod). Zkopírovanou URL vložte do otevřeného requestu v aplikaci Postman do pole **Enter URL.**



Obrázek 26 – Vložení URL do requestu

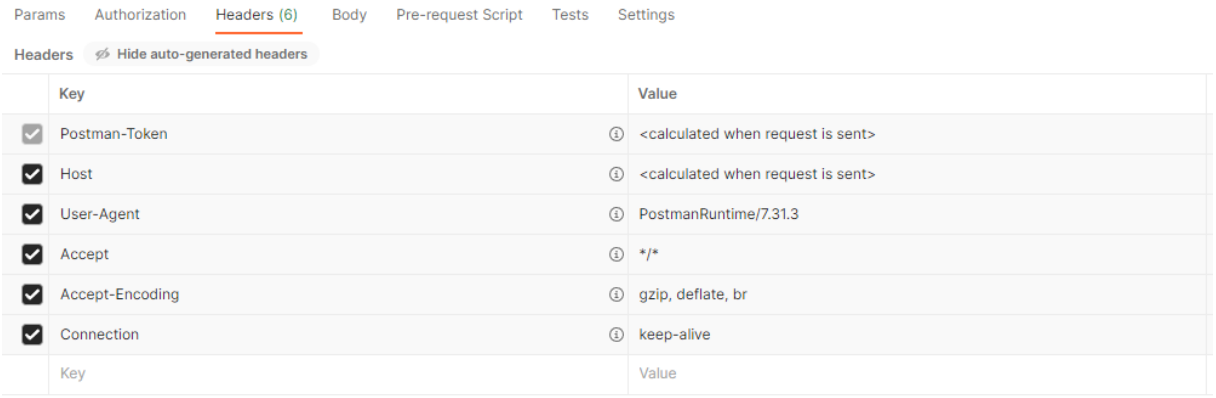
Zdroj: [1]

Z dokumentace je zřejmé, že pro získání informací o astronomickém snímku je třeba zvolit metodu GET. Tato metoda je při otevření nového requestu defaultně zvolena, a proto ji v tomto

případě nemusíte vybírat z listu nabízených metod. V případě, že by byla vyžadována jiná metoda pro daný request (například POST nebo PUT), na tomto místě ji musíte změnit.

Postman automaticky předvyplní headers requestu, které poskytují informace serveru o klientovi. Pro zobrazení nebo skrytí headers je nutné přejít na záložku **Headers** a kliknout na ikonu ve tvaru oka. Předvyplněné headers jsou defaultně skryté a nelze je měnit. V případě potřeby změny hodnoty některé z defaultně předvyplněných headers je nutné zadat stejný název do sloupce **Key** a novou hodnotu vložit do pole **Value**.

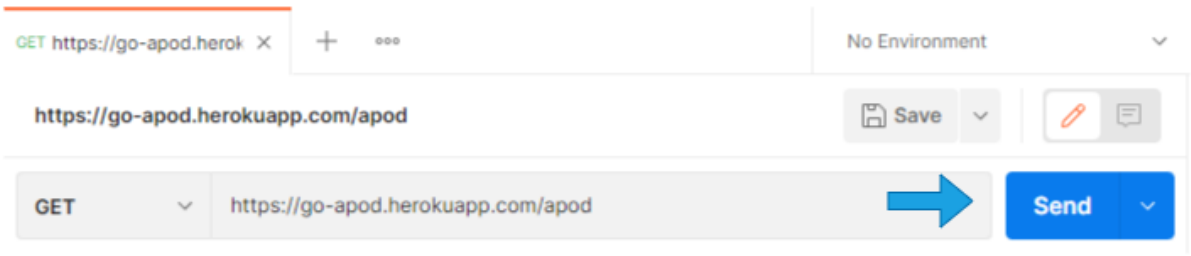
V našem případě není potřeba headers upravovat.



Obrázek 27 – Headers requestu

Zdroj: [1]

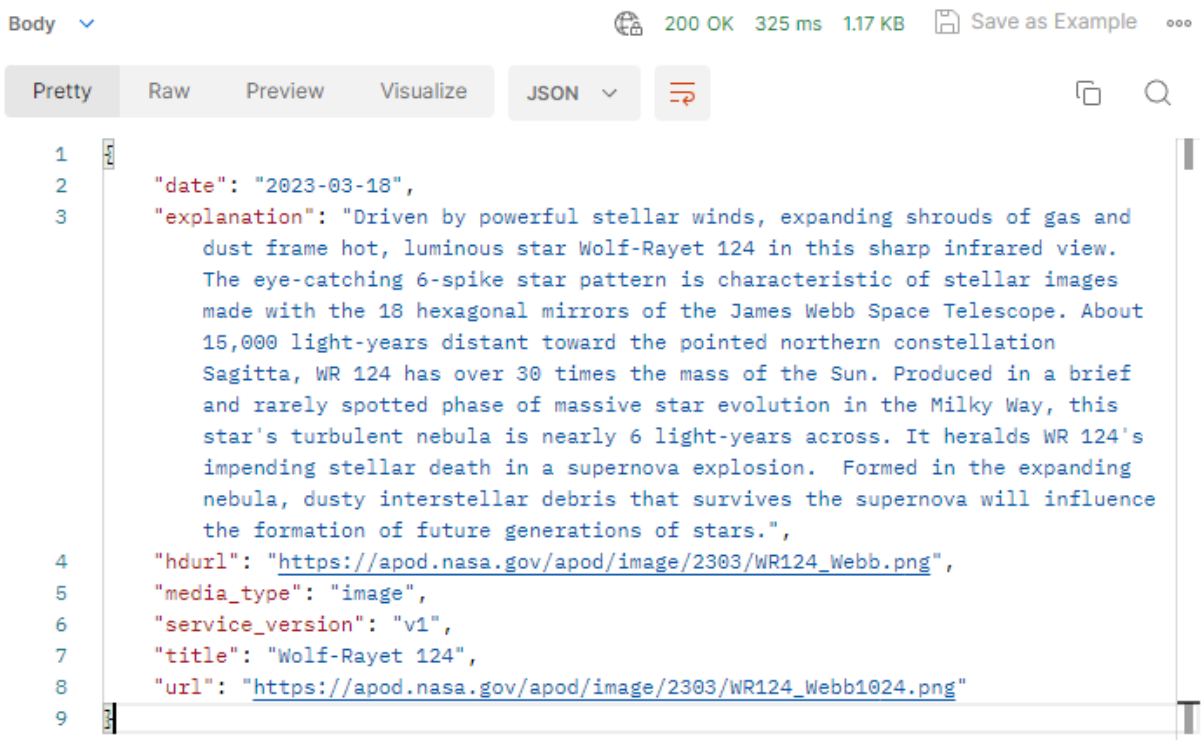
Nyní je již request připravený a pro jeho odeslání klikněte na tlačítko **Send**.



Obrázek 28 – Odeslání requestu

Zdroj: [1]

Po odeslání requestu API vrátila odpověď. Pokud vše proběhlo v pořádku, byl navrácen stavový kód 200 OK a tělo odpovědi ve formátu JSON. Tělo odpovědi obsahuje veškeré informace, které daný endpoint poskytuje. Z těla odpovědi lze zjistit například, jaký je popis astronomického snímku dne nebo odkaz na snímek, viz obrázek č. 9.



Obrázek 29 – Odpověď requestu ve formátu JSON

Zdroj: [1]

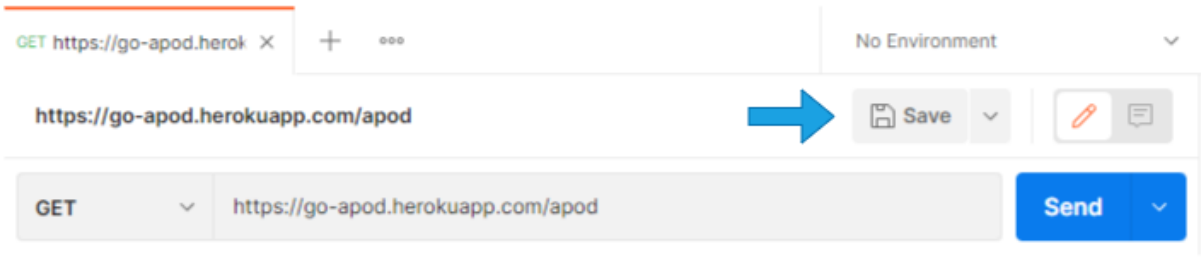
Úkol:

Najděte v dokumentaci pro API **Astronomy Picture of the Day** endpoint pro zobrazení snímku dne a odešlete request. Odpověď si zobrazte ve formátu Preview.

4 PRÁCE S COLLECTIONS

V předchozí kapitole jste si vyzkoušeli vytvořit a odeslat request. Aby byla práce s API přehlednější, jednodušší a efektivnější, je vhodné si requesty organizovat. Pro organizaci requestů v aplikaci Postman slouží collections. Pro následující příklad využijeme vytvořený request z předchozí kapitoly.

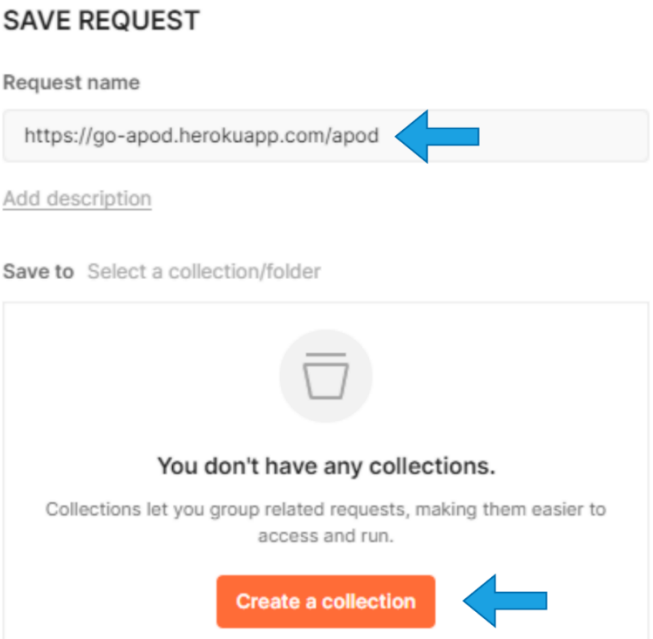
U daného requestu klikněte na tlačítko **Save**.



Obrázek 30 – Ukládání requestu

Zdroj: [1]

Následně se otevře okno vyzývající k doplnění informací pro uložení requestu. Nastavte název requestu v poli **Request name**, který bude vhodně vyjadřovat daný request. Poté klikněte na tlačítko **Create a collection**.

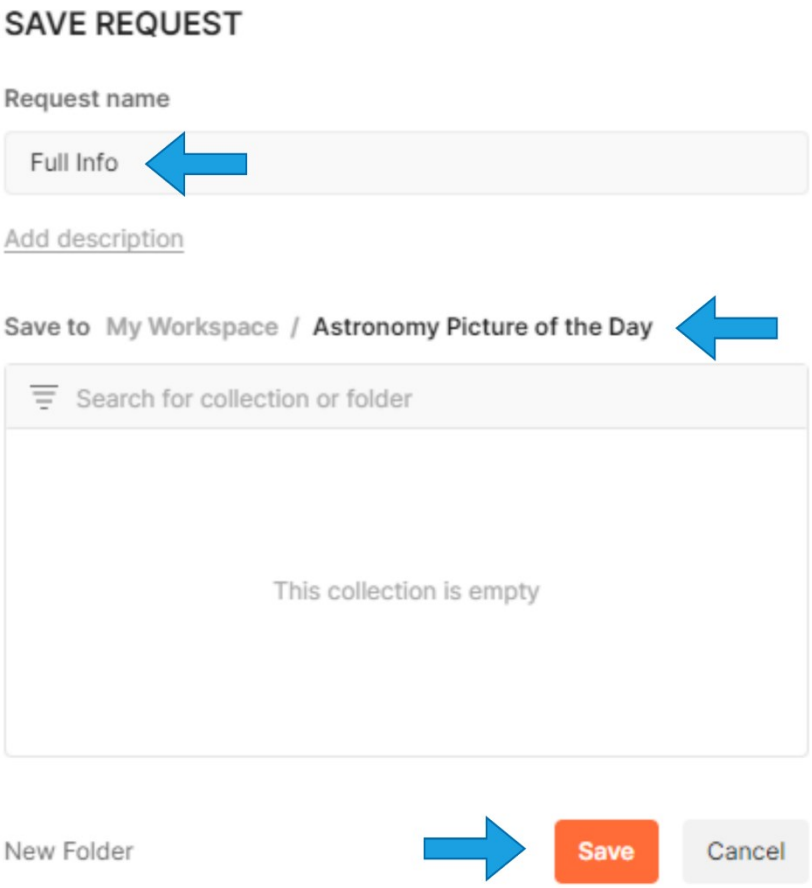


Obrázek 31 – Vytváření nové collection

Zdroj: [1]

Nastavte název collection. Do collection budeme ukládat všechny requesty k této API, proto ji pojmenujte například podle názvu API.

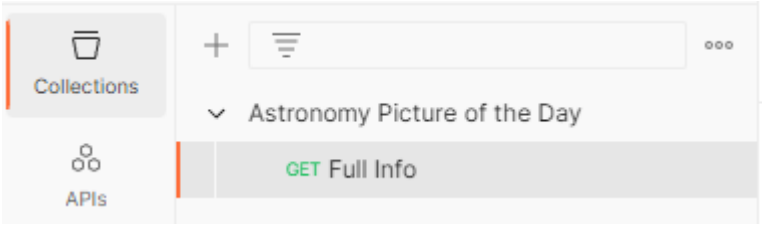
Collection ponese název Astronomy Picture of the Day. Poté request uložte kliknutím na tlačítko **Save**.



Obrázek 32 – Přiřazení requestu do collection

Zdroj: [1]

Po uložení byste měli najít request v left sidebaru v záložce **Collections** v nově vytvořené collection Astronomy Picture of the Day.



Obrázek 33 – Zobrazení uloženého requestu v collection

Zdroj: [1]

Postman neprovádí automatické ukládání, proto po jakékoli sebemenší změně, kterou v requestu provedete, musíte následně vše manuálně uložit.

Tip:

Práci si můžete usnadnit s využitím zkratek, které najdete v **Settings** v záložce **Shortcuts**. Jednotlivé zkratky si lze dle potřeb předdefinovat. Například pro uložení requestu můžete použít zkratku CTRL+S.

Úkol:

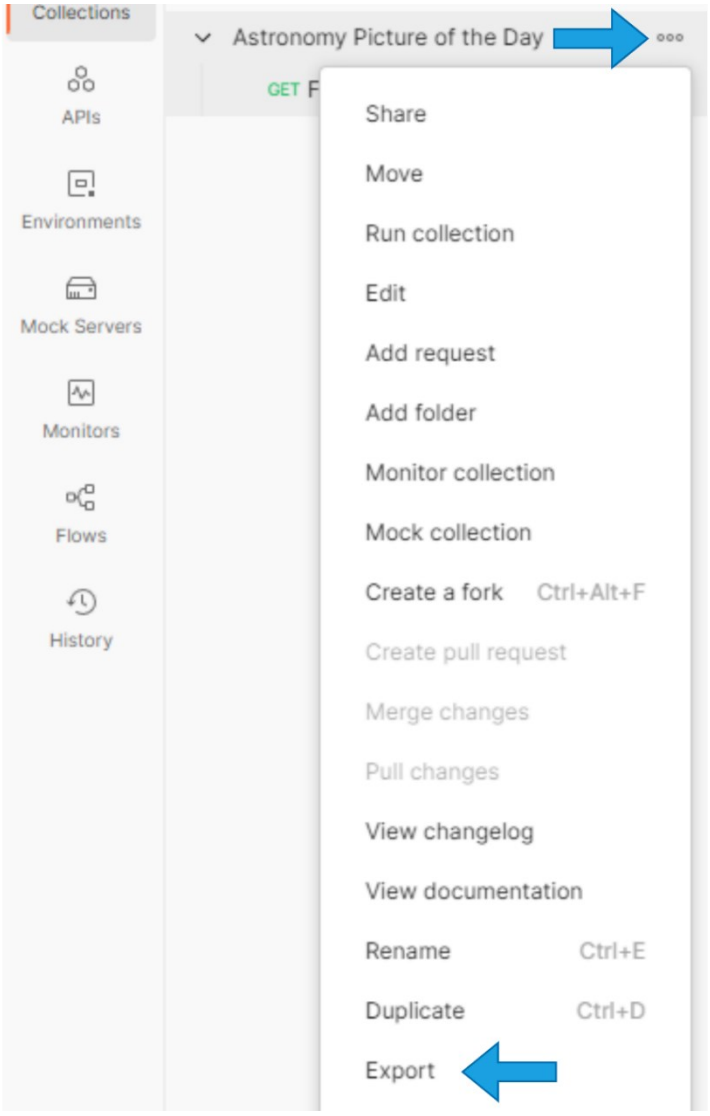
Vytvořte nový request s URL https://go-apod.herokuapp.com/image, request pojmenujte a uložte do vytvořené collection Astronomy Picture of the Day.

5 EXPORTOVÁNÍ, IMPORTOVÁNÍ A SDÍLENÍ COLLECTIONS

Postman nabízí možnost sdílet, exportovat a importovat vytvořené collections, což usnadňuje přístup dalším uživatelům, kteří mají zájem s danou collection pracovat. Následující podkapitoly popisují, jak collection exportovat, importovat nebo sdílet.

5.1 Export collection

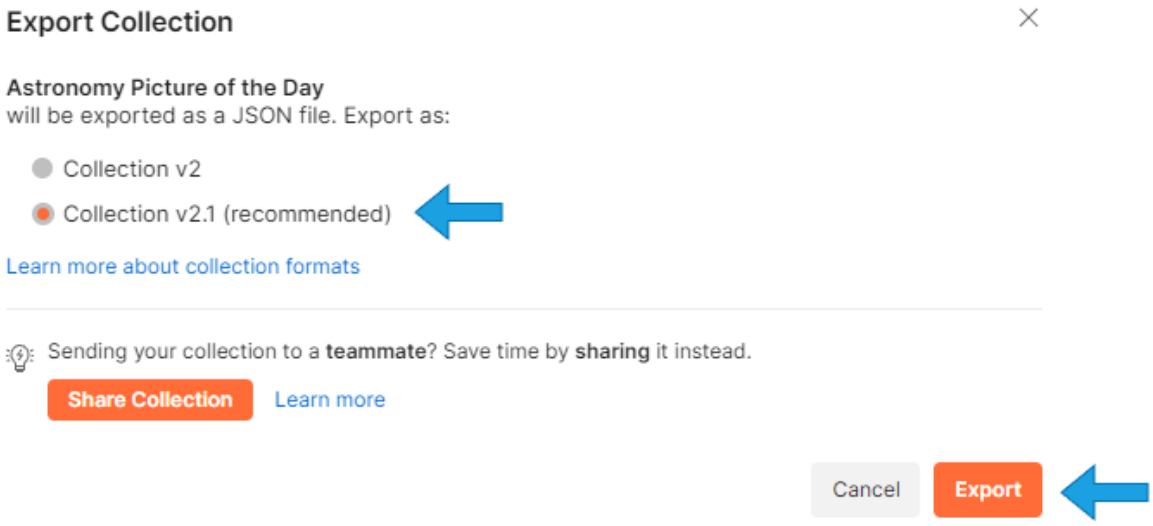
Pro export collection najeďte kurzorem na collection, kterou chcete exportovat a klikněte na tlačítko **View more actions** (3 horizontální tečky). Objeví se vám nabídka akcí. Z nabídky klikněte na tlačítko **Export**.



Obrázek 34 – Collection – View more actions

Zdroj: [1]

Před tím, než collection vyexportujete, musíte vybrat, v jaké verzi collection chcete export provést. Vyberte doporučovanou verzi souboru JSON (v tomto případě Collection v2.1) a klikněte na tlačítko **Export**.



Obrázek 35 – Export collection do JSON souboru

Zdroj: [1]

JSON soubor si uložte. Pro zajímavost si můžete prohlédnout strukturu JSON souboru.

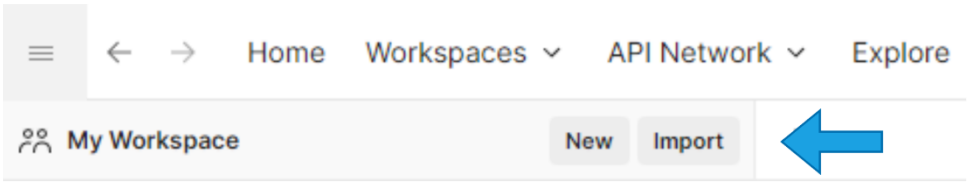
Obrázek 36 – Struktura exportovaného JSON souboru

Zdroj: [1]

Exportovaný soubor si můžete uschovat nebo ho můžete přeposlat libovolnému uživateli, který si ho následně může naimportovat do svého workspace.

5.2 Import collection

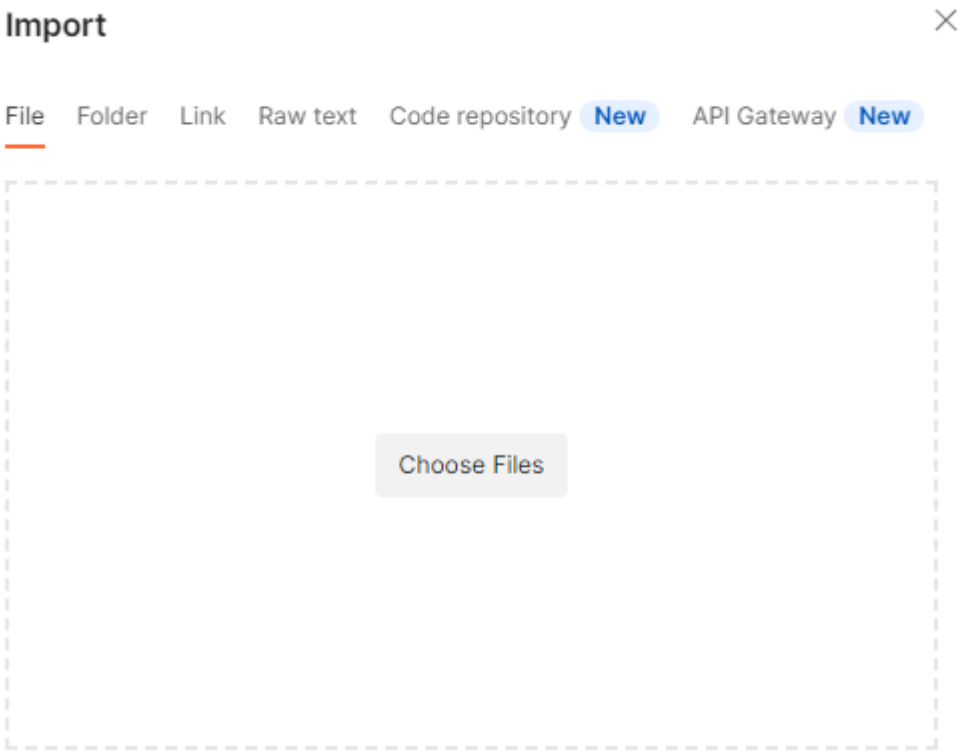
Pro import JSON souboru vyberte workspace, do kterého chcete collection naimportovat a klikněte na tlačítko **Import**.



Obrázek 37 – Tlačítko Import

Zdroj: [1]

Z nabídky vyberte záložku **File** a tlačítkem **Choose Files** vyberte exportovaný JSON soubor z předchozí kapitoly.

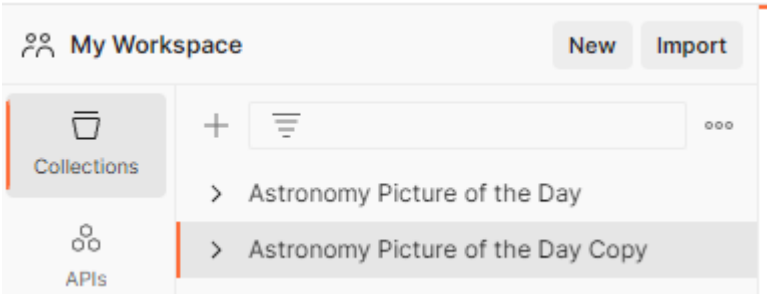


Obrázek 38 – Vkládání souboru k importu

Zdroj: [1]

Po vybrání souboru klikněte na tlačítko **Import** a poté **Import as Copy**.

Pokud import proběhl v pořádku, v seznamu collections uvidíte importovanou collection.



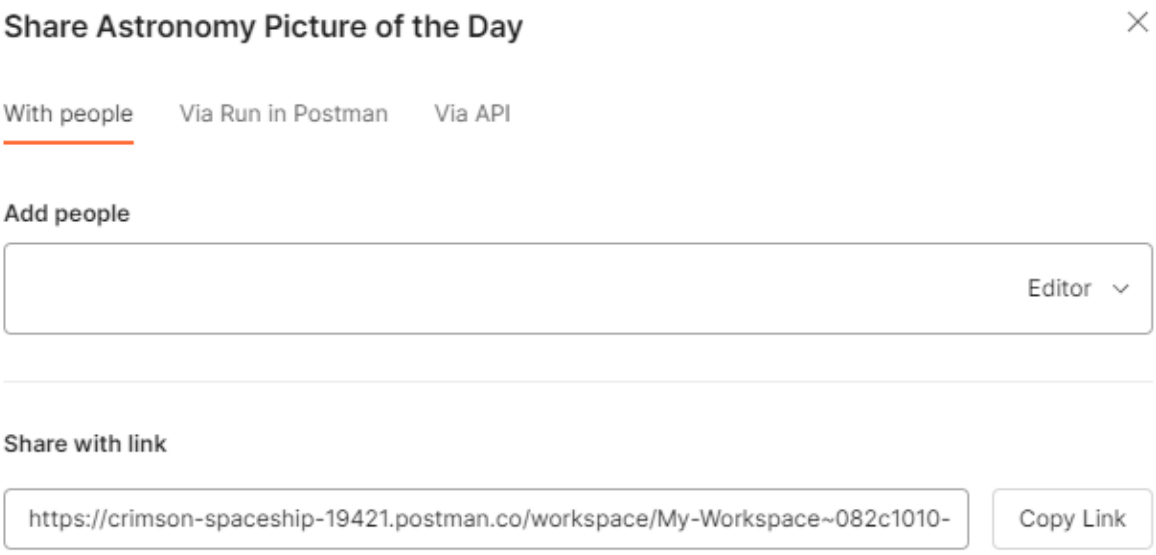
Obrázek 39 – Import Collection do My Workspace

Zdroj: [1]

Jelikož není potřeba mít dvě stejné collections, klikněte znovu na **View more actions** a z nabídky možností klikněte na tlačítko **Delete**, čímž dojde ke smazání importované kopie.

5.3 Sdílení collection

Collections lze také sdílet s jinými uživateli přímo z workspace. U dané collection klikněte na **View more actions** a z nabídky možností klikněte na tlačítko **Share**.



Obrázek 40 – Nabídka možností ke sdílení Collection

Zdroj: [1]

Pro sdílení lze využít více možností:

- Sdílení přes odkaz,

- Odeslání pozvánky konkrétním uživatelům na e-mail,
- Sdílení přes Run in Postman,
- Sdílení přes API.

Podrobnější popis jednotlivých variant sdílení je k dispozici na stránce:

https://learning.postman.com/docs/collaborating-in-postman/sharing/

Velmi podobným způsobem se dají importovat, exportovat a sdílet celá API, jednotlivé requesty nebo prostředí.

Úkol:

Vyhledejte si volně dostupnou API a naimportujte ji do svého workspace.

6 PRÁCE S PROMĚNNÝMI

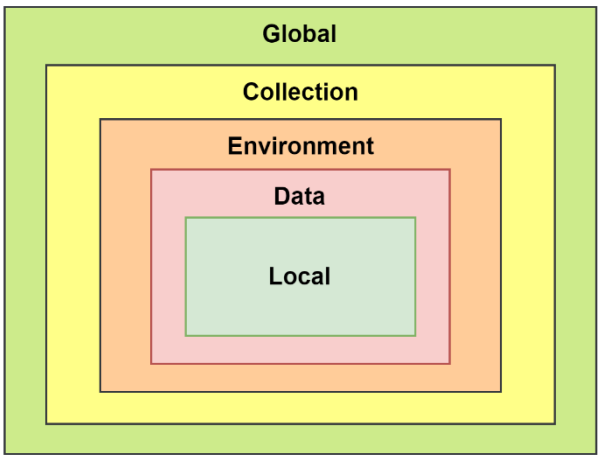
Proměnné umožňují uživatelům definovat a ukládat hodnoty. To znamená, že se jako tester můžete vyhnout ručnímu zadávání stejných hodnot znovu a znovu, což zrychluje a zjednodušuje práci.

Postman rozlišuje mezi několika druhy proměnnými. Jedná se o proměnné:

1. **Global variables** – globální proměnné, které jsou k dispozici v rámci daného workspace.
2. **Collection variables** – collection proměnné k dispozici uvnitř collection, tyto proměnné jsou nezávislé na prostředí.
3. **Environment variables** – proměnné prostředí jsou platné ve zvoleném prostředí.
4. **Data variables** – jedná se o datové proměnné, které nabývají hodnoty pouze z CSV nebo JSON souborů a jsou využitelné například při spuštění data-driven testů v Collection Runneru.
5. **Local variables** – lokální proměnné platné v rámci requestu, jsou tvořené při skriptování. [6]

Pokud je definována proměnná, která je platná pro více oblastí, například jako collection variable i data variable, je v requestu použita vždy hodnota proměnné z nižší úrovně. V tomto případě to bude hodnota data variable.

Na obrázku č. 21 jsou zobrazeny úrovně jednotlivých typů proměnných. [7]



Obrázek 41 – Úrovně proměnných v aplikaci Postman

Zdroj: vlastní zpracování dle [7]

Ukážeme si práci s proměnnými na následujícím příkladu, kdy hodnota proměnné Forename je definována zároveň jako collection variable i local variable.

Použijeme volně dostupnou API z https://nationalize.io/, která odhaduje národnost člověka podle křestního jména.

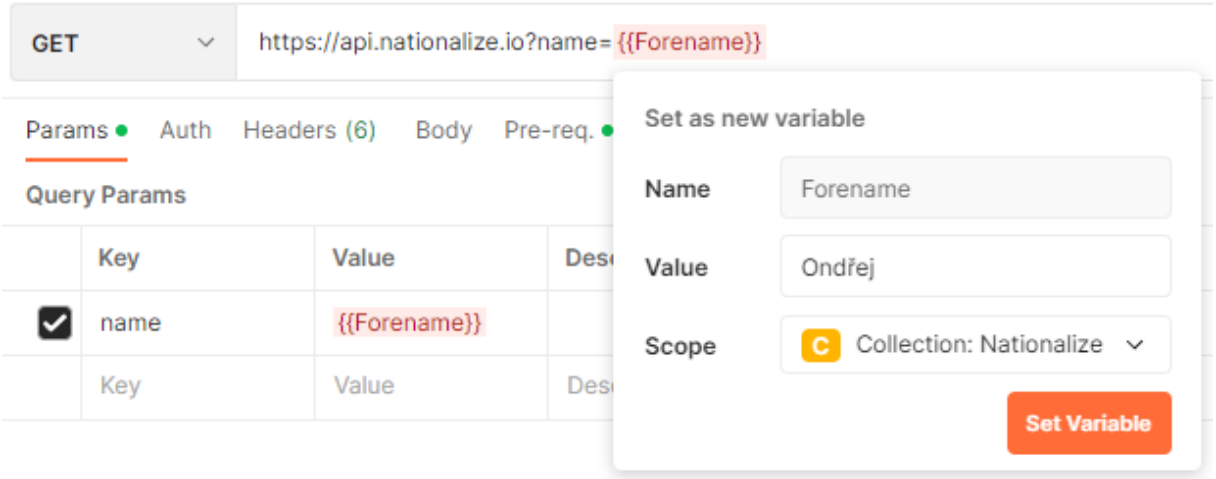
Vytvořte v aplikaci Postman nový HTTP request a vložte do něj následující URL.

URL: https://api.nationalize.io?name=michael

Request uložte do nové collection. Jako proměnnou si nastavte hodnotu pro query parametr name. Hodnotu tohoto query parametru nahraďte proměnnou Forename.

Výsledná URL bude: https://api.nationalize.io?name={{Forename}}

Po najetí kurzorem na proměnnou v URL je proměnná označená jako Unresolved variable. Klikněte na tlačítko **Add new variable**. Nastavte hodnotu proměnné v poli **Value** na libovolné jméno a v poli **Scope** vyberte Collection. Poté klikněte na tlačítko **Set Variable**.



Obrázek 42 – Vytváření Collection proměnné

Zdroj: [1]

Nastavení collection variable si můžete ověřit po kliknutí na collection v záložce **Variables**.

Dále nastavíme local variable. V requestu, se kterým pracujete překliknete do záložky **Pre-req.**

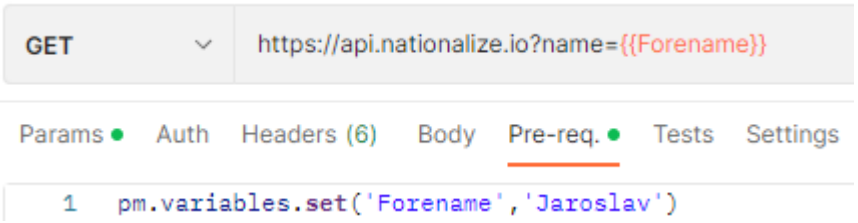
Poznámka:

Pre-request script je proveden před odesláním requestu a používá se například k nastavování proměnných nebo k validaci dat v reqeustu.

V textovém poli nastavte hodnotu local variable Forename na jiné jméno, než máte nastavené u collection variable. Local variable vytvoříte například pomocí následujícího skriptu:

pm.variables.set('Forename','Jaroslav')

Obrázek č. 23 zobrazuje, jak by měla vypadat URL requestu obsahující proměnnou a umístění skriptu pro nastavení local variable.



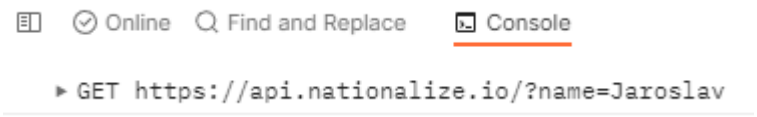
Obrázek 43 – Vytváření local variable

Zdroj: [1]

Pokud máte nastavené collection i local variable, tlačítkem **Send** odešlete request.

V levém dolním rohu aplikačního rozhraní otevřete **Console** a ověřte, která hodnota proměnné byla v request odeslána.

Z obrázku vidíte, že v URL requestu je hodnota proměnné Jaroslav, tedy hodnota definovaná v lokální proměnné.



Obrázek 44 – Hodnota query parametru odeslaného requestu

Zdroj: [1]

Tip:

Console je vhodné používat při debuggingu, tj. v situaci, kdy při odeslání requestu nastane chyba. Informace obsažené v konzoli mohou pomoci zjistit, zda je chyba v API, nebo chyba vznikla například překlepem v URL při vytváření requestu.

Úkol:

Nastavte v aplikaci proměnnou Forename jako global variable a zároveň jako environment variable s různými hodnotami proměnných. Určete, která proměnná bude v requestu odeslána. Request následně odešlete a ověřte hodnotu v URL s pomocí nástroje Console.

7 PŘÍPRAVA TESTOVACÍCH SKRIPTŮ PRO TESTOVÁNÍ API

Pro psaní testovacích skriptů v aplikaci Postman se používá programovací jazyk Javascript. Základní principy, jak psát testovací skripty a jak probíhá jejich exekuce, budou znázorněny v následujících podkapitolách.

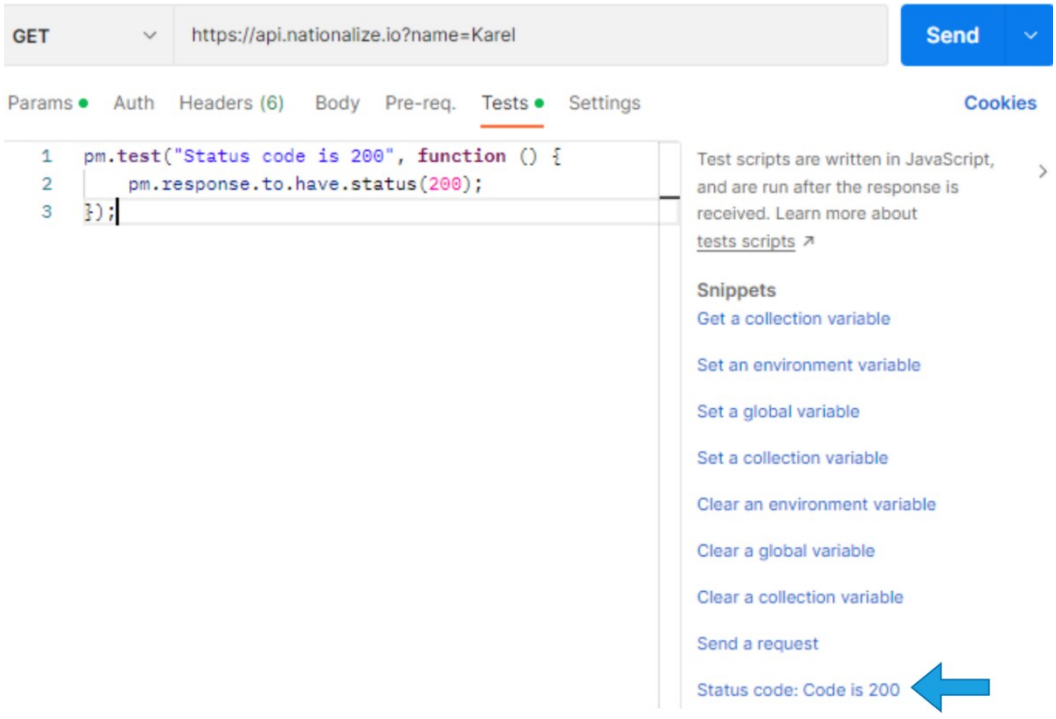
7.1 Testy na stavové kódy

V rámci funkčního testování tester ověřuje, zda API funguje, jak má, a vrací správné odpovědi definované v požadavcích. Aby tester nemusel manuálně kontrolovat každý request zvlášť, lze si připravit vlastní testy, které budou automaticky kontrolovat stavové kódy samy a tester si pouze zobrazí výsledky exekuce. Ke psaní testů slouží záložka **Tests**.

Pro následující příklad opět využijeme API s URL https://api.nationalize.io?name=michael. Klikněte na záložku **Tests** a vložte následující skript, případně využijte předpřipravený snippet z nabídky snippetů.

```
pm.test("Status code is 200", function () {
    pm.response.to.have.status(200);
});
```

Testovací skript a jeho umístění je zobrazeno na obrázku 25.



Obrázek 45 – Vkládání testovacích skriptů v záložce Tests

Zdroj: [1]

Poznámka:

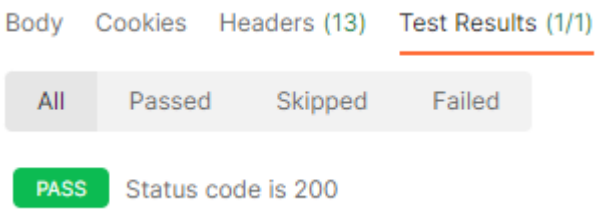
Snippety obsahují nejčastěji používané skripty pro testování API.

Když se podíváme na zápis tohoto skriptu, vidíme, že je použita knihovna **pm** a metoda **test.** Následuje název testu v uvozovkách a funkce.

Lze si všimnout, že kód skriptu je velmi dobře čitelný. Není složité rozpoznat, že účelem tohoto testu je zkontrolovat, jestli vrácený stavový kód obsahuje hodnotu 200. Testy mohou být zpravidla poměrně dobře čitelné, jelikož Postman pro psaní asercí používá knihovnu Chai Assertion – https://www.chaijs.com/api/bdd/.

Tato knihovna dává skriptům větší srozumitelnost a kód je tak čitelnější i pro další zainteresované osoby, které s danými skripty budou případně pracovat, ale tyto skripty nevytvářely. Pro ověření, zda test projde odešlete request kliknutím na tlačítko **Send**.

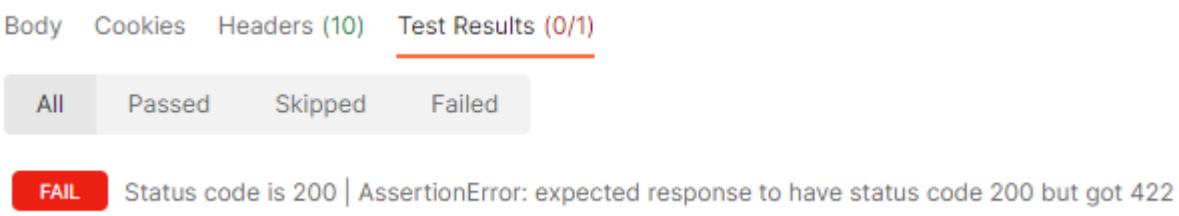
Poté překlikněte na záložku **Test Results** a zkontrolujte, zda test prošel.



Obrázek 46 – PASS – pozitivní výsledek testu

Zdroj: [1]

Test s názvem **Status code is 200** prošel, má hodnotu PASS. Pro ověření, zdali test bude fungovat i v případě, že API nevrátí stavový kód 200, upravte URL tak, aby API vrátilo jinou hodnotu (například změňte název query parametru name) a odešlete request znovu.



Obrázek 47 – Negativní výsledek testu

Zdroj: [1]

Výsledek testu je FAIL, jelikož API vrátila status kód 422 – Unprocessable Entity z důvodu chybějícího parametru name.

Z dokumentace používané API vyčteme, že je možné v rámci requestu poslat až 10 parametrů. Na základě této informace zhotovíme dva testovací skripty na okrajové hodnoty, které budou ověřovat vrácené stavové kódy. Jeden request bude v URL obsahovat dohromady 10 parametrů name, zatímco druhý request bude obsahovat 11 parametrů name. Pro efektivnější práci s jednotlivými requesty si přidejte část URL před parametry do collection proměnné. Jednotlivé requesty uložte do collection a vhodně je pojmenujte.

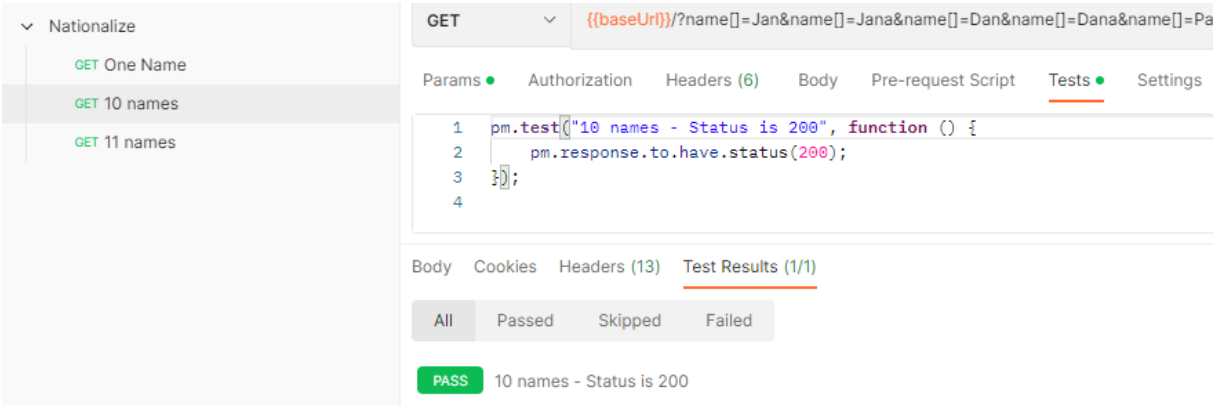
Vzorová URL pro request s 10 parametry:

{{baseUrl}}/?name[]=Jan&name[]=Jana&name[]=Dan&name[]=Dana&name[]=Pavel&name[]=Pavla&name[]=Martin&name[]=Martina&name[]=Karel&name[]=Karla

Očekávaný stavový kód vrácený API je 200.

Jakmile budete mít request připravený, odešlete ho.

Po odeslání requestu vidíte, že test prošel.



Obrázek 48 – Výsledek testu requestu s názvem 10 names

Zdroj: [1]

Tip:

Pokud používáte identický test pro více requestů, lze test vložit přímo ke collection nebo k jejím podsložkám. Test bude i tak vždy vykonán po spuštění requestu a v případě nutnosti změny takového testu postačí provést úpravu pouze na jednom místě.

Vzorová URL pro request s 11 parametry:

{{baseUrl}}/?name[]=Jan&name[]=Jana&name[]=Dan&name[]=Dana&name[]=Pavel&name[]=Pavla&name[]=Martin&name[]=Martina&name[]=Karel&name[]=Karla&name[]=Oliver

Očekávaný stavový kód vrácený API je 422.

Obrázek 49 – Výsledek requestu 11 names

Zdroj: [1]

Z výsledku testů vidíte, že i tento test prošel.

Úkol:

Napište a proveďte test pro ověření, že API vrátí správný status kód v případě, že je v URL requestu nevalidní API key.

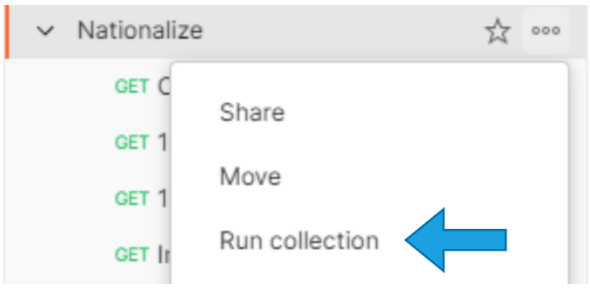
Očekávaný stavový kód vrácený API je 401.

URL: https://api.nationalize.io?name=peter&apikey={YOUR_API_KEY}

7.2 Collection Runner

V předchozí kapitole jste si připravili několik různých requestů v rámci jedné collection. Abyste nemuseli každý request posílat manuálně a zvlášť, nabízí Postman funkci Collection Runner, která automaticky pošle všechny vybrané requesty ve stanoveném pořadí.

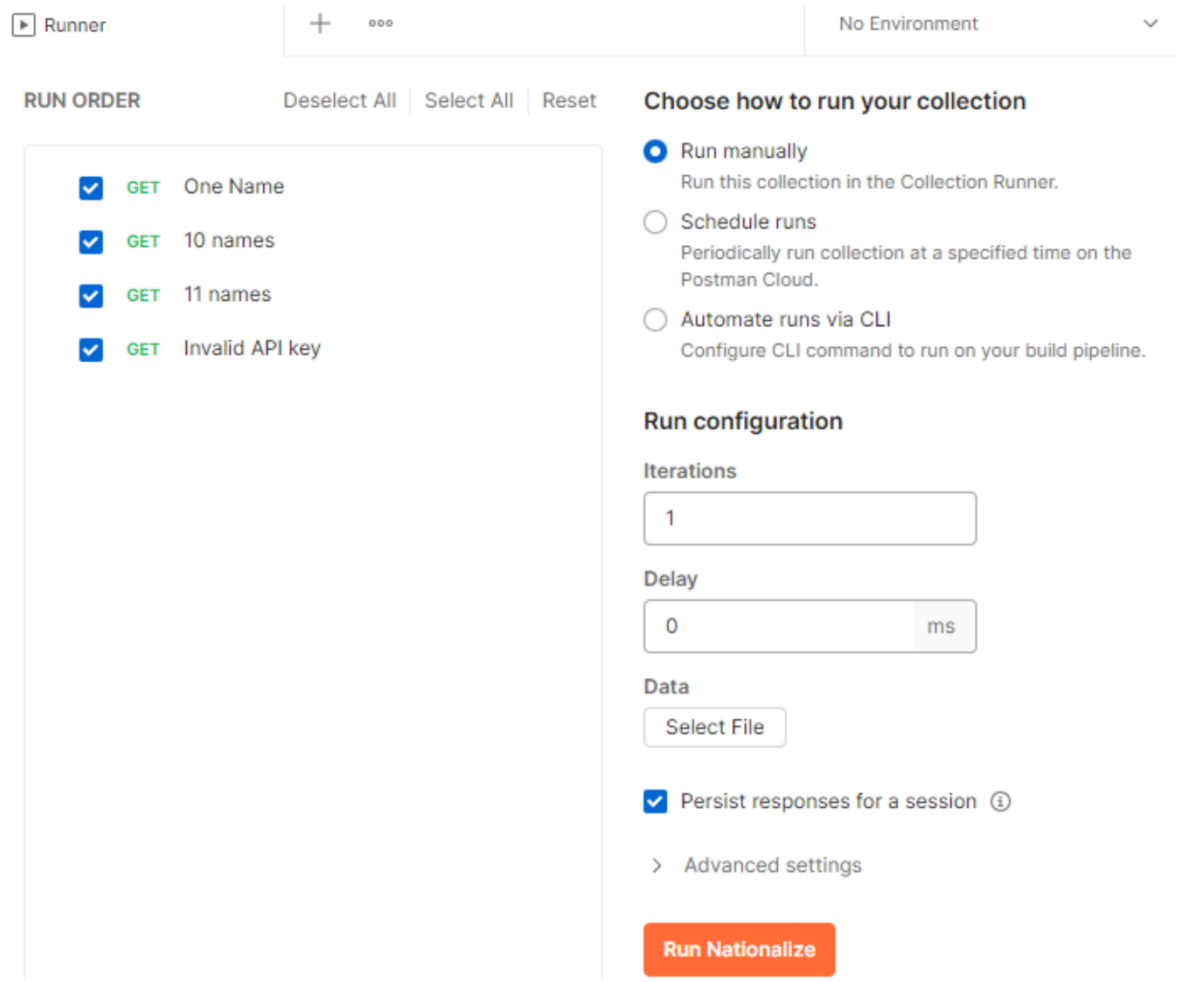
U dané collection klikněte na **View more actions** a z nabídky možností vyberte **Run Collection**.



Obrázek 50 – Run collection

Zdroj: [1]

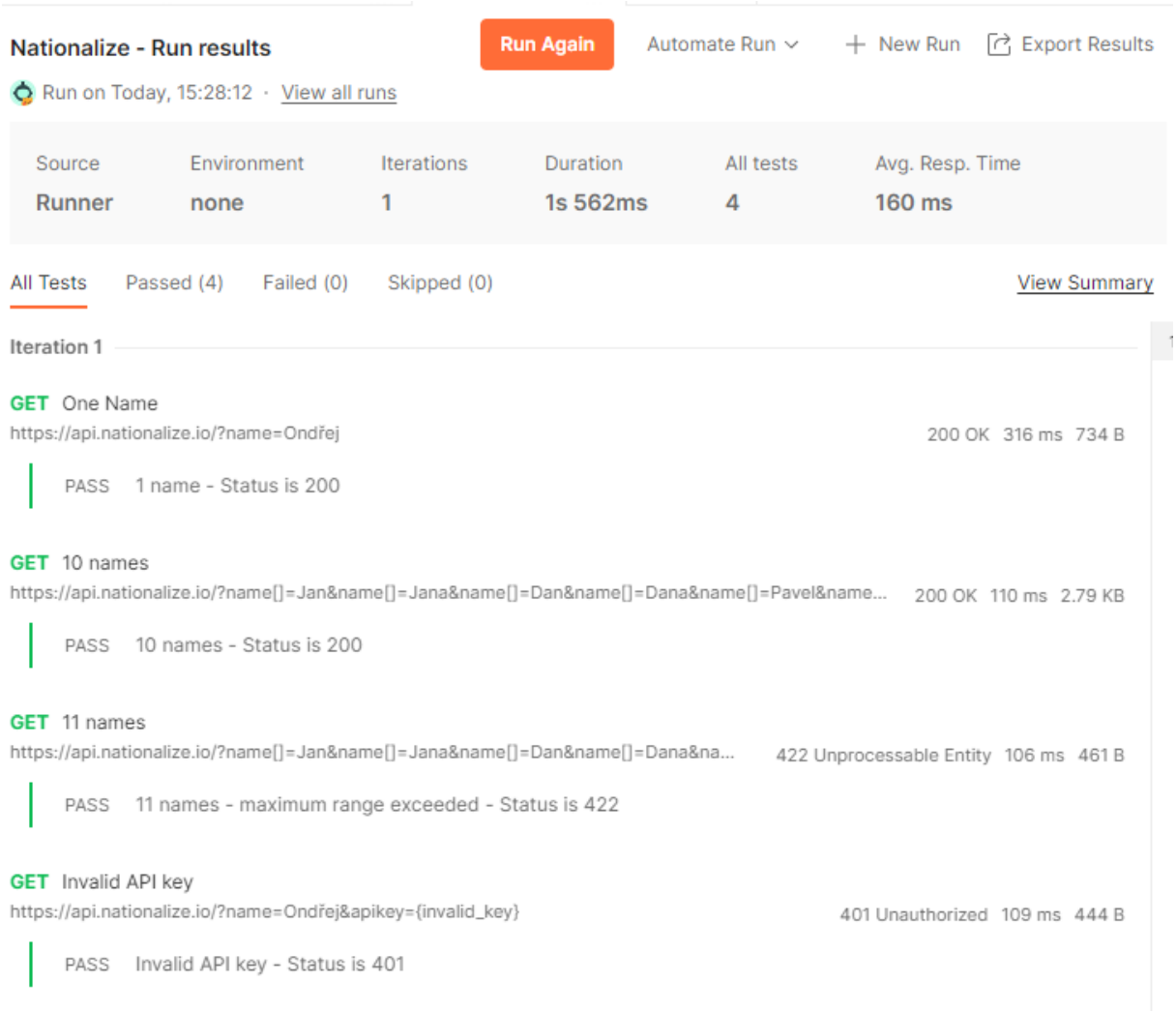
Po kliknutí se otevře Runner. V části **RUN ORDER** můžete zvolit, které requesty chcete do běhu zahrnout a případně upravit pořadí requestů. V konfiguraci Runneru si také můžete zvolit počet iterací, tj. kolikrát chcete vybraný seznam requestů odeslat. Pokud chcete zobrazit výsledky včetně response jednotlivých requestů, zaškrtněte checkbox **Persist responses for a session**. Poté klikněte na tlačítko **Run Nationalize**.



Obrázek 51 – Nastavení manuálního běhu Collection Runneru

Zdroj: [1]

Po spuštění se zobrazí výsledky běhu. Z výsledků je zřejmé, že všechny testy úspěšně prošly, proběhla jedna iterace a celková doba exekuce trvala 1 s 562 ms.



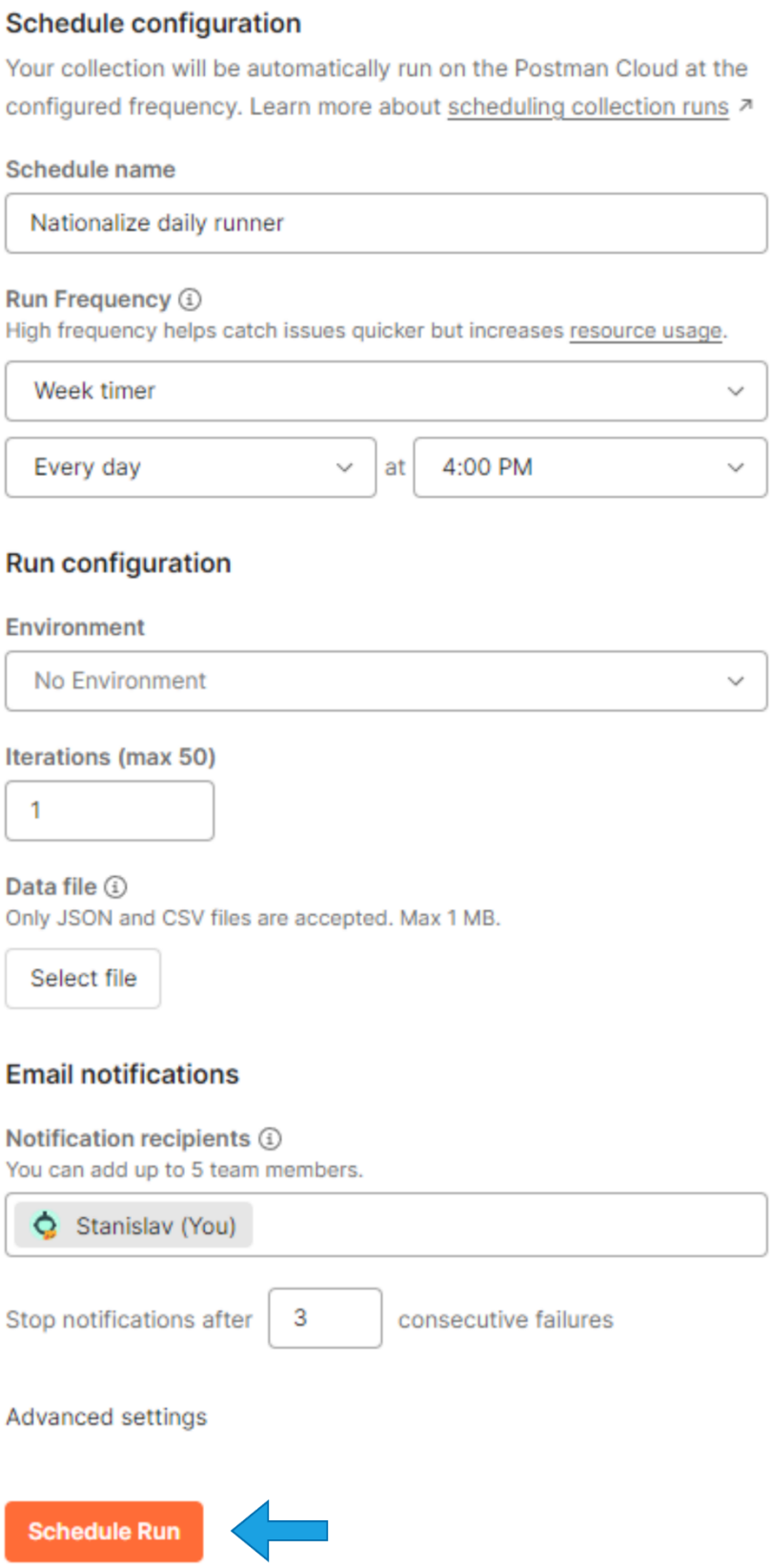
Obrázek 52 – Výsledky testů Collection Runneru

Zdroj: [1]

7.3 Monitoring API

Collection Runner lze také automatizovat tak, aby běžel pravidelně ve stanovený čas. U dané Collection klikněte znovu na tlačítko **Run Collection** a místo defaultně vybrané možnosti **Run collection manually** vyberte **Schedule Runs.** Nastavte název plánu, interval, počet iterací a klikněte na tlačítko **Schedule Run.**

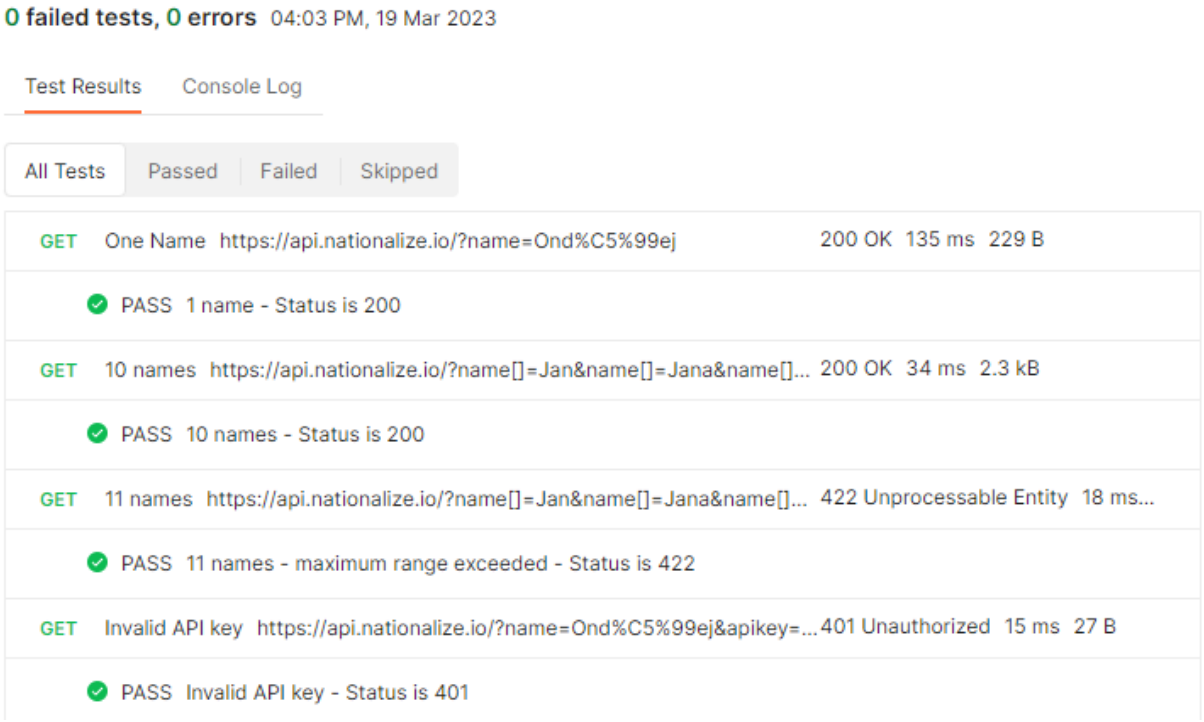
Vzorové nastavení plánu je zobrazené na obrázku č. 33.



Obrázek 53 – Plánování pravidelného běhu testů

Zdroj: [1]

Poté vyčkejte na nastavený čas a zkontrolujte běh. Ten najdete v Collection v záložce **Runs**. Výsledky zobrazíte kliknutím na tlačítko **View** u daného běhu. Z výsledků vidíme, že v dané iteraci všechny testy úspěšně prošly.



Obrázek 54 – Výsledky naplánovaných testů

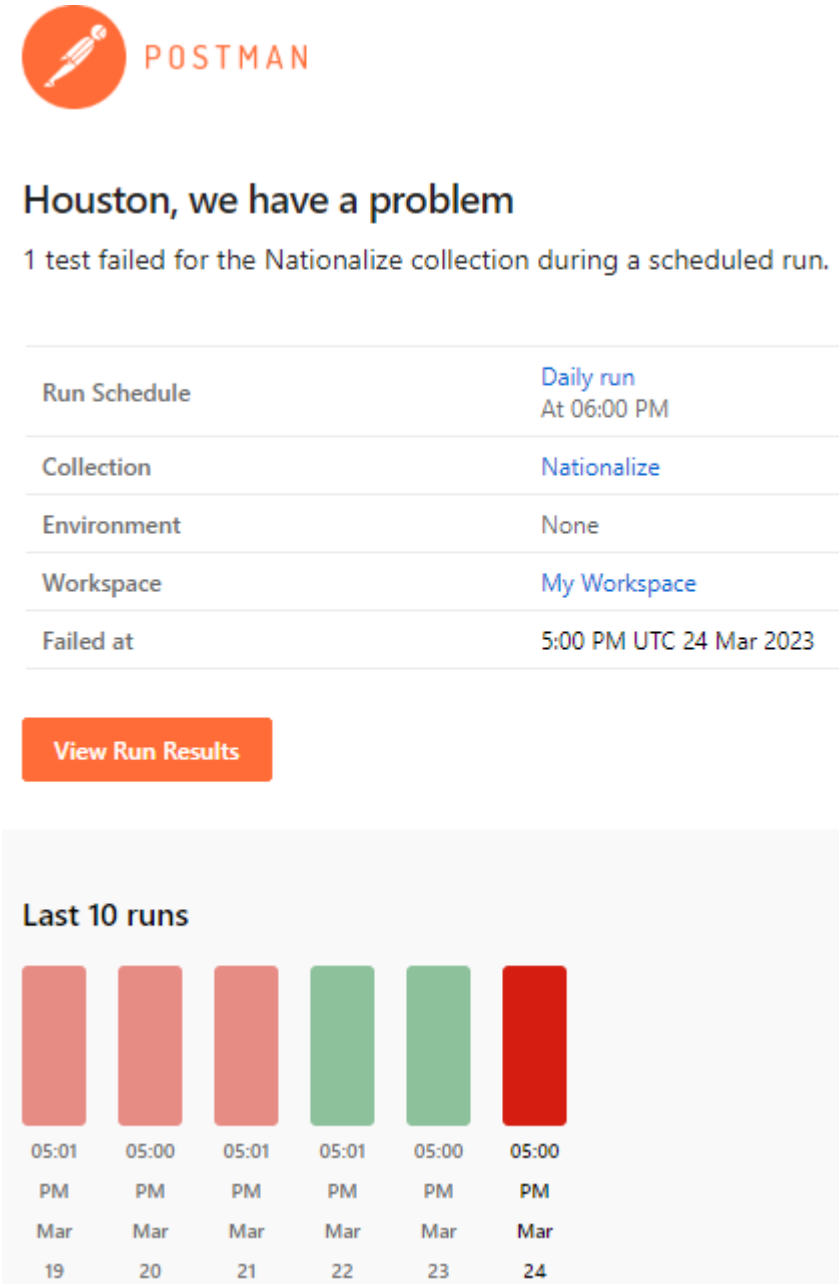
Zdroj: [1]

Tímto způsobem lze zajistit automatizaci testů, které budou vykonávány pravidelně ve stanovených intervalech. Zvolení uživatelé budou informováni e-mailem v případě, že se v běhu vyskytne chyba.

Pokud žádné chyby nenastanou, v podstatě se o dané testy uživatelé nemusí více starat, o všechnu práci se postará sám Postman.

Na obrázku č. 35 je zobrazena e-mailová notifikace, kterou uživatel obdrží na svůj e-mail v případě, že v exekuci testů nastala chyba.

Notifikace obsahuje informace o tom, v jaký den a v kolik hodin chyba nastala a které collection a workspace se týká. Z e-mailu se lze prokliknout přímo na podrobnější výsledky exekuce.



Obrázek 55 – E-mailová notifikace oznamující chybu

Zdroj: [1]

Úkol:

Vytvořený automatický běh smažte a vytvořte nový monitoring, který bude spouštět řadu requestů každý den každé dvě hodiny.

7.4 Testování těla odpovědi

V této kapitole bude prezentováno, jak otestovat hodnoty vrácené v odpovědi.

Navážeme na request připravený v kapitole 7.1.

URL: https://api.nationalize.io?name=Michael.

Test bude ověřovat, zda hodnota query parametru name byla vrácena ve stejné podobě v těle odpovědi.

Z nabídky testovacích skriptů vyberte snippet **Response body: JSON value check.** Modifikujte název testu na "Name value in response body equals query parameter value".

Posléze upravte funkci pm.expect takovým způsobem, aby provedla kontrolu, že klíč name bude mít hodnotu query parametru.

Výsledný skript by mohl vypadat například takto:

```
pm.test("Name value in response body equals query parameter value", function () {
    var jsonData = pm.response.json();
    pm.expect(jsonData.name).to.eql('Michael');
});
```

Jakmile budete mít vše připravené, odešlete request kliknutím na tlačítko **Send**.

Test skončil PASS, pokud se hodnota query parametru rovnala hodnotě v těle odpovědi.

Na obrázku č. 36 je zobrazen výsledek vykonaného vytvořeného testu.

Obrázek 56 – Výsledek testu těla odpovědi

Zdroj: [1]

V těle odpovědi má klíč name hodnotu Michael, test byl vyhodnocen správně.


```
{
    "country": [
        {
            "country_id": "AT",
            "probability": 0.064
        },
        {
            "country_id": "DE",
            "probability": 0.059
        },
        {
            "country_id": "DK",
            "probability": 0.058
        },
        {
            "country_id": "IE",
            "probability": 0.051
        },
        {
            "country_id": "AU",
            "probability": 0.047
        }
    ],
    "name": "Michael"
}
```


Úkol:

Nahraďte parametr name proměnnou a upravte testovací skript takovým způsobem, aby správně ověřil hodnotu vrácenou API v odpovědi.

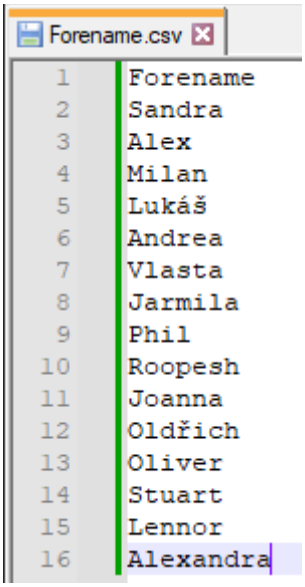
8 DATA-DRIVEN TESTOVÁNÍ

Postman nabízí možnost provádět data-driven testy, kdy lze spouštět request s různými vstupními daty. Na následujícím příkladu bude znázorněno, jak data-driven testy včetně přípravy dat provádět. Využijeme vytvořený request s připravenými skripty z kapitoly 7.4.

URL: https://api.nationalize.io?name={{Forename}}

Vytvořte CSV soubor s daty, kdy v prvním řádku bude název proměnné, která je nastavena místo hodnoty query parametru name. V dalších řádcích již nastavte pouze hodnoty parametru name. V souboru můžete připravit libovolný počet hodnot proměnné Forename. Počet hodnot se následně bude rovnat počtu iterací v Collection Runneru. Soubor uložte.

Na obrázku č. 37 je znázorněn příklad vstupních dat ve formátu CSV.



Obrázek 57 – Vstupní data pro data-driven testování

Zdroj: [1]

V aplikaci Postman si otevřete Collection Runner a s pomocí tlačítka **Choose file** vložte vytvořený CSV soubor s daty.

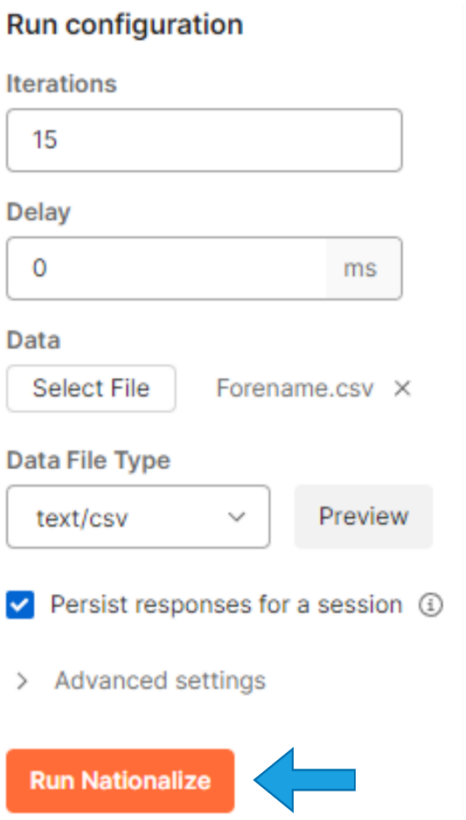
Následně si s pomocí tlačítka **Preview** můžete prohlédnout, zda se vám povedlo naimportovat správně všechna vstupní data. Vidíte, že pro každou iteraci je připravena jiná hodnota proměnné Forename.

Na obrázku 38 je zobrazena náhled testovacích dat.

Obrázek 58 – Náhled testovacích dat

Zdroj: [1]

Tlačítkem **Run Nationalize** spusťte běh viz obrázek č. 39.



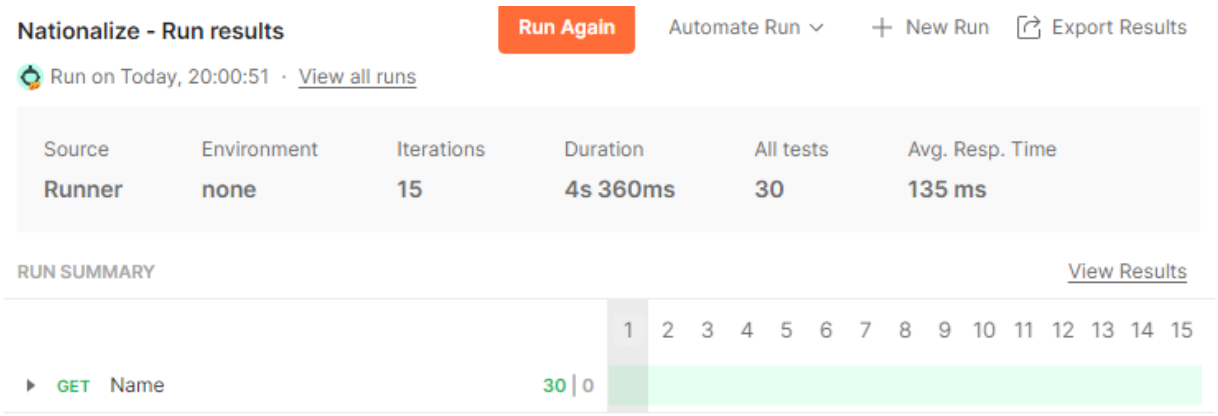
Obrázek 59 – Spuštění data-driven testů

Zdroj: [1]

Počkejte na dokončení všech iterací a zobrazte výsledky. Z výsledků běhu vidíme, že proběhlo 15 iterací daného requestu s různými daty dle CSV souboru. Jelikož v každé iteraci byly provedeny dva testy, výsledkem je 30 úspěšně vykonaných testů.

Celková doba trvání byla 4 s a 360 ms.

Podrobné výsledky data-driven testů jsou zobrazeny na obrázku č. 40.



Obrázek 60 – Výsledky data-driven testů

Zdroj: [1]

9 NEWMAN

Newman je nástroj umožňující automatizovat collections s pomocí příkazové řádky. V této kapitole bude na příkladu znázorněno, jakým způsobem lze spustit collection v Newmanu.

Instalace aplikace Newman:

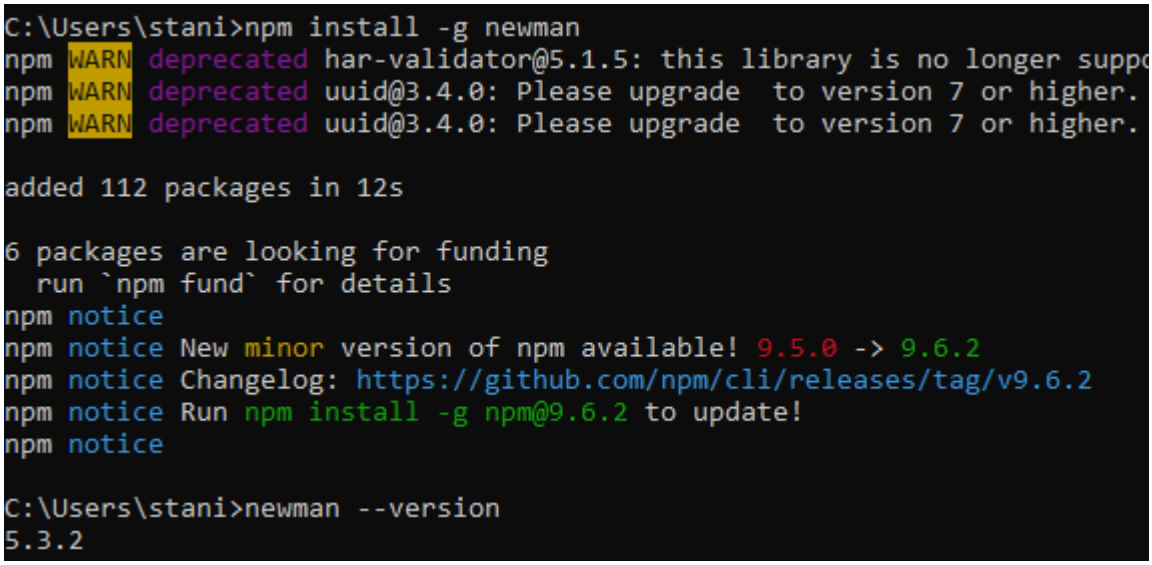
Pro spuštění aplikace Newman je nutné nainstalovat Node.js. Nainstalujte si správnou verzi dle Vašeho operačního systému.

Odkaz ke stažení Node.js – https://nodejs.org/en/download

Po instalaci Node.js stačí spustit příkazovou řádku a nainstalovat newman následujícím příkazem: npm install -g newman.

Poznámka:

Nainstalovanou verzi si lze ověřit příkazem newman --version.



Obrázek 61 – Instalace Newman

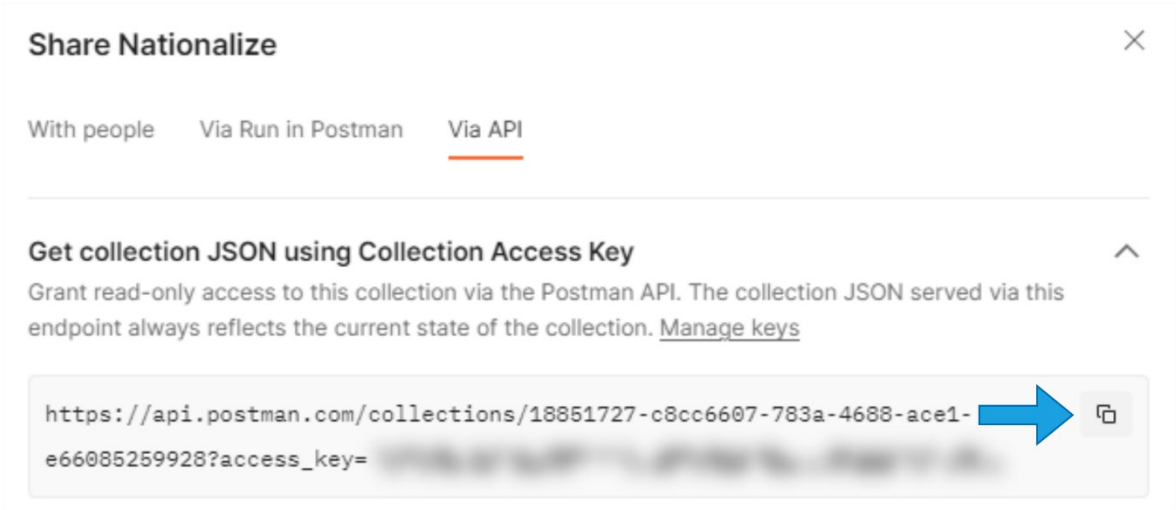
Zdroj: vlastní zpracování v příkazové řádce

Poté vyberte collection, kterou budete chtít v Newmanu spustit a klikněte na **View more actions.**

Z nabídky možností klikněte na tlačítko **Share.**

Přesuňte se na záložku **Via API** a vygenerujte si pro daný link access key viz obrázek č. 42.

Poté zkopírujte URL pro danou collection včetně vygenerovaného access key.



Obrázek 62 – Generování access key ke collection

Zdroj: [1]

S pomocí příkazu *newman run {{vygenerované URL pro collection}}* spusťte v příkazové řádce exekuci collection a vyčkejte na výsledky testů. Následně Newman vypíše odeslané requesty a v přehledné tabulce zobrazí výsledky.

Na obrázku č. 43 jsou zobrazené výsledky běhu collection. Vidíme, že dva prerequest skripty a jedna aserce selhaly.

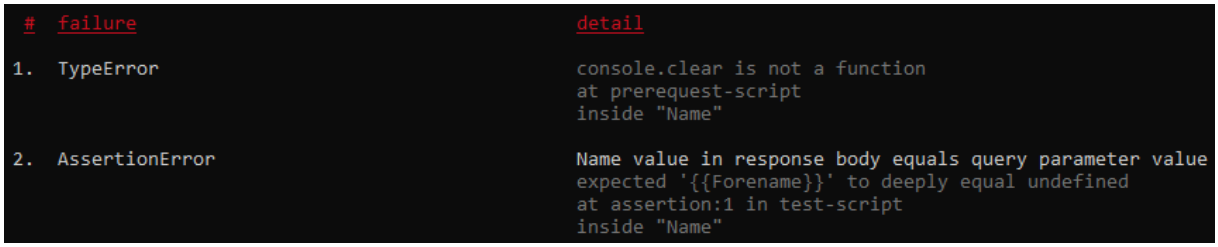
	executed	failed
iterations	1	0
requests	4	0
test-scripts	8	0
prerequest-scripts	6	2
assertions	5	1
total run duration: 1285ms		
total data received: 2.38kB (approx)		
average response time: 204ms [min: 115ms, max: 469ms, s.d.: 152ms]		

Obrázek 63 – Newman – výsledky testů

Zdroj: vlastní zpracování v příkazové řádce

V případě, že vše nešlo podle plánů, Newman v příkazové řádce popíše také důvody selhání.

Na obrázku č. 44 je ukázka popisu důvodů selhání z posledního běhu.



Obrázek 64 – Newman – popis selhání

Zdroj: vlastní zpracování v příkazové řádce

Newman nezná uložené proměnné a prostředí, které jsou nastavené, pokud spouštíte collection z aplikace Postman. V případě, že jsou v collection použity a chcete collection spustit v aplikaci Newman, je nutné je definovat v příkazu a odkázat na ně s pomocí JSON souboru, který bude tato data obsahovat.

Pokud v Postmanu provedete jakékoliv změny u vybrané collection, musíte aktualizovat URL odkazující na tuto collection nebo opět vyexportovat JSON soubor.

Tip:

Seznam možných příkazů, které lze použít v příkazové řádce v aplikace Newman, najdete na stránce:

https://learning.postman.com/docs/collections/using-newman-cli/newman-options/

Úkol:

Exportujte collection do JSON souboru a spusťte ji v příkazové řádce příkazem: newman run {{název JSON souboru}}.

POUŽITÁ LITERATURA

[1] GHOST_ICON, c2010-2023. Open book free icon. In: *Flaticon* [online]. Freepik Company [cit. 2023-04-15]. Dostupné z: https://www.flaticon.com/free-icon/open-book_2406774

[2] SMALLLIKEART, c2010-2023. Homework free icon. In: *Flaticon* [online]. Freepik Company [cit. 2023-04-15]. Dostupné z: https://www.flaticon.com/free-icon/homework_2113781

[3] EUCALYP, c2010-2023. Study free icon. In: *Flaticon* [online]. Freepik Company [cit. 2023-04-15]. Dostupné z: https://www.flaticon.com/free-icon/study_3212114

[4] FREEPIK, c2010-2023. Introduction free icon. In: *Flaticon* [online]. Freepik Company [cit. 2023-04-15]. Dostupné z: https://www.flaticon.com/free-icon/introduction_4917018

[5] Postman. Postman for Windows. Version 10.9.4 [software]. [cit. 2023-04-11]. Dostupné z: https://www.postman.com/downloads/. Požadavky na systém: Windows 64-bit; velikost 165 MB.

[6] Using variables: Variable scopes, 2023. *Postman: Postman, Inc.* [online]. San Francisco: Postman [cit. 2023-02-24]. Dostupné z: https://learning.postman.com/docs/sending-requests/variables/

[7] DESPA, Valentin. Demystifying Postman Variables: HOW and WHEN to use Different Variable Scopes. *Medium* [online]. 07. 03. 2019 [cit. 2023-03-25]. Dostupné z: https://medium.com/apis-with-valentine/demystifying-postman-variables-how-and-when-to-use-different-variable-scopes-66ad8dc11200